PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-232779
(43)Date of publication of application : 02.09.1998

(51)Int.Cl. G06F 9/38
G06F 9/30
G06F 9/45

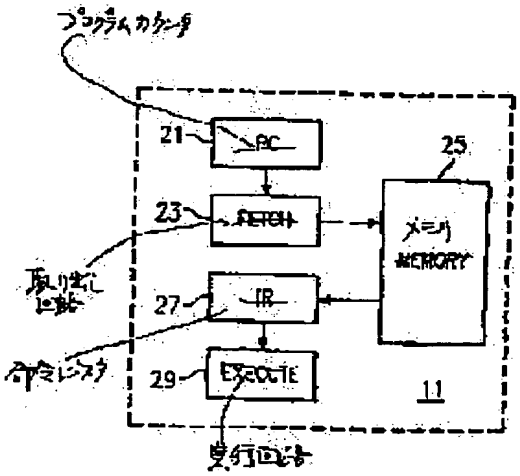
(21)Application number : 10-048480
(22)Date of filing : 23.01.1998

(71)Applicant : TEXAS INSTR INC <TI>
(72)Inventor : SIMAR LAURENCE R
SESHAN NATARAJAN
TATGE REID E
DAVIS ALAN L

(30)Priority
Priority number : 97 36223 Priority date : 24.01.1997 Priority country : US

(54) METHOD AND DEVICE FOR PARALLEL INSTRUCTION PROCESSING

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a data processing system with which one part of instructions is checked and it can be designated whether the instructions are to be executed parallelly or successively.
SOLUTION: A data processing circuit 11 has a circuit (program counter 21, program memory 23 and memory 25) for generating one set of instructions and one part of each instruction includes the designation showing whether the other instruction in one set of instructions and that instruction can be simultaneously executed or not. A program execution circuit 29 receives one set of instructions, selectively responds to one part of instructions and simultaneously executes the plural instructions shown by that one part of instructions.



LEGAL STATUS
[Date of request for examination]
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]

[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号

特開平10－232779

(43)公開日　平成10年(1998)９月２日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/38	３７０	Ｇ０６Ｆ	9/38	３７０Ｘ
	9/30	３５０		9/30	３５０Ｆ
	9/45			9/44	３２２Ｆ

審査請求　未請求　請求項の数３　ＯＬ　外国語出願　（全 48 頁）

(21)出願番号　特願平10－48480

(22)出願日　平成10年(1998)１月23日

(31)優先権主張番号　０３６２２３

(32)優先日　1997年１月24日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　590000879
テキサス　インスツルメンツ　インコーポレイテツド
アメリカ合衆国テキサス州ダラス，ノース　セントラルエクスプレスウエイ　13500

(72)発明者　ローレンス　アール．シマー
アメリカ合衆国　テキサス州リッチモンド，モートン　リーグ　ロード　2103

(72)発明者　ナタラジャン　セスハン
アメリカ合衆国　テキサス州ヒューストン，エラ　リー　レーン　9550　ナンバー　408

(74)代理人　弁理士　浅村　皓　（外３名）

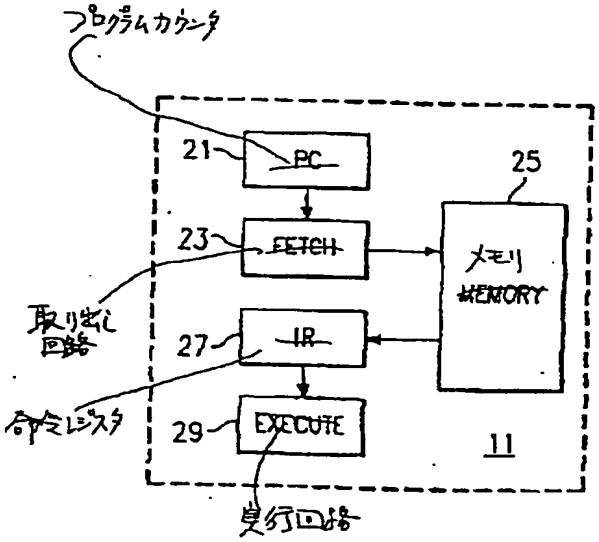
最終頁に続く

(54)【発明の名称】　命令並列処理方法及び装置

(57)【要約】

【課題】　命令の一部を検査して、命令を並列実行するか逐次実行するかを指定できるデータ処理システムを開示する。

【解決手段】　データ処理回路１１には、１組の命令を発生させる回路（プログラムカウンタ２１，プログラムメモリ２３，メモリ２５）があり、各命令の１部分に、その１組の命令の中の別の命令と、その命令とを同時に実行できるか否かを示す指定が含まれている。プログラム実行回路２９は、その１組の命令を受信し、命令の１部分に選択的に応答して、その命令の１部分によって示される複数の命令を同時に実行する。



【特許請求の範囲】
【請求項１】　それぞれの命令を同時に実行することができるか否かを示す、それぞれの命令の１部分を含む１組の命令を発生する回路を有するデータ処理回路を含むデータ処理システムであって、
前記データ処理回路は、前記発生回路に接続され、前記１組の命令を受信するプログラム実行回路を含みかつ前記命令の１部分に選択的に応答して前記命令の１部分によって示される複数の前記命令を同時に実行するデータ処理システム。
【請求項２】　データ処理システムにおいてプログラム命令を処理する方法であって、
その命令の中の別の命令と同時にそれぞれの命令を実行できるか否かを示すそれぞれの命令の１部分を、１組の命令に供給するステップと、
複数の前記命令を同時に実行できるか否かを、前記命令の１部分から決定するステップと、を含む方法。
【請求項３】　データ処理システムによって実行するため、プログラムをコンパイルする方法であって、
プログラムの中の第１のプログラム命令の直後に続く第２のプログラム命令と同時に、その第１のプログラム命令を実行できるか否かを決定するステップと、
第１の命令を第２の命令と同時に実行できるか否かを示す命令の１部分を、その第１の命令に供給するステップと、を具備する方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、一般にデータ処理に関し、より詳細には、プログラム命令の並列実行および逐次実行の両方を備えたデータ処理に関する。
【０００２】
【発明が解決しようとする課題】データ処理システムおよびデータプロセッサは、実際上生活のあらゆる側面に影響を与える多数の用途で使用されている。関連する処理システムおよびデータプロセッサの速度およびスループットを速くすることにより、これらの多数の用途の有用性を向上させることができるのが普通である。
【０００３】速度およびスループットを向上させることが可能な場合の１つの方法は、プログラム命令を逐次的に実行するよりむしろ並列に実行することである。この点に関する公知の方法の１つは、プログラム命令の並列実行または逐次実行を指定する特殊なモードの命令を使用することである。この点に関する公知の別の方法は、並列型パケットにヌル命令を指定するためにマスクを使用することである。この点に関する公知の別の解決方法は、並列モードだけでデータ処理を実行することである。
【０００４】前述の手法は速度およびスループットを改善することができるが、実現することがかなり難しいために望ましくない上、処理に伴うオーバーヘッドの点で費用がかかるという欠点がある。
【０００５】したがって、上に説明した方法に付随する実現上の困難と、処理に伴うオーバーヘッドとが減少するように、プログラム命令を並列で実行できるようにすることが望ましい。
【０００６】本発明の目的は、与えられたプログラム命令の一部を利用して、その命令を他のプログラム命令と同時に実行することができるか否かを決定することである。
【０００７】
【課題を解決するための手段】一般に、そしてまた本発明の形式においては、１組の命令を発生する回路をもつデータ処理装置が用意されており、これらの命令には、それぞれ該当する命令を、その複数の命令の中の別の命令と同時に実行できるか否かを示す命令の１部分が含まれている。このデータ処理回路には、前記発生回路に接続されていて、その１組の命令を受信すると、インジケータ部に応答して複数の命令を同時に実行するプログラム実行回路が含まれている。
【０００８】本発明の別の形式では、データ処理装置の中の中央処理装置（ＣＰＵ）を動作させる方法は、それぞれ該当する命令を同時に実行できるか否かを示すため、１組の命令にそれぞれの命令の１部分を与えるステップと、その命令の１部分から複数の命令を同時に実行できるか否かを決定するステップとから構成されている。
【０００９】本発明の別の実施例は、説明および図面から明らかになるであろう。
【００１０】
【発明の実施の形態】添付の図面において、異なる図面および表における対応する番号と符号は、とくに断りのない限り対応する部品を示している。
【００１１】図１は、本発明によるデータ処理システム１０のブロック図である。データ処理システム１０にはデータ処理回路１１および周辺回路１３，１５，１７，１９が含まれている。図１の代表的実施例において、データ処理回路１１は、データ処理回路1.1と周辺回路１３，１５，１７，１９との間で情報を転送するため、周辺回路１３，１５，１７，１９のそれぞれに接続されている。しかし、以下の説明から明らかになるとおり、本発明によるデータ処理システムは、この説明の前で後でも当業者ならば想像できるように、相互に接続されかつデータ処理回路１１に接続されるいかなる量の、また（たとえば周辺装置１３，１５，１７，１９など）いかなる形式の周辺回路および周辺装置が含まれていてもよい。
【００１２】図２は、図１のデータ処理回路１１の１つの代表的実施例の一部を示している。図２において、取り出し回路２３が、プログラムカウンタ２１によって指定されるアドレスでメモリ２５にアクセスすると、命令

パケットがそのアドレスで命令レジスタ２７にロードされる。プログラム実行回路２９は、命令レジスタ２７に保持されているその命令パケットの中の複数の命令を復号化して実行する。

【００１３】図３は、メモリ２５から取り出された命令パケットの基本フォーマットを示している。開示した代表的実施例においては、命令パケットには４個の３２ビット命令Ａ，Ｂ，Ｃ，Ｄが含まれている。図３に示すように、命令Ａ，Ｂ，Ｃ，Ｄはメモリ２５の連続したアドレスに格納されている。したがって、正常のプログラム命令の逐次実行中、命令Ａが最初に実行され、命令Ｂ，Ｃ，Ｄが順番に続いて実行される。

【００１４】図３の各命令のビット０はｐ－ビットと呼ばれている。ｐ－ビットは、命令の実行方法を定義する。図３の命令パケットのｐ－ビットは、プログラム実行回路２９によって左から右に検査される。ある命令のｐ－ビットが論理１に等しい場合、そのパケットの中の次の順番の命令が、最初に言及した命令と同時に実行される。グラム実行回路２９は、その命令パケットの中のｐ－ビットが論理０に等しい命令に到達するまでこの規則を適用する。

【００１５】ある命令のｐ－ビットが０の場合、その命令の後に（そしてその命令と並列に実行されるすべての命令の後に）続いて次の順番の命令が実行される。プログラム実行回路２９は、命令パケットの中のｐ－ビットが論理１の命令に到達するまでこの規則を適用する。

【００１６】図４から図１９は、上に説明したｐ－ビットの規則の応用例を示している。

【００１７】図４は、すべてのｐ－ビットが０の命令パケットを示している。したがって、命令Ａから命令Ｄは図５に示すとおり、順番に連続して実行される。

【００１８】図６は、命令Ａ，Ｂ，Ｃのｐ－ビットが１に等しく、命令Ｄのｐ－ビットが０の命令パケットを示している。したがって、命令Ａ，Ｂ，Ｃ，Ｄは同時に実行される。すなわち、図７に示すとおり並列で同時に実行される。

【００１９】図８の命令においては、命令Ｃのｐ－ビットだけが１にセットされているので、図９の実行順序になる。すなわち、命令Ａ，Ｂが逐次実行され、並列に実行される命令Ｃ，Ｄが、その後に続いて実行される。

【００２０】図システム１０の命令パケットにおいては、命令Ｂのｐ－ビットだけが１にセットされているので、図１１の実行順序になる。すなわち、命令Ａが実行され、それに続いて命令Ｂ，Ｃが並列に実行され、その後に続いて命令Ｄが実行される。

【００２１】図１２の命令パケットにおいては、命令Ｂ，Ｃのｐ－ビットが１にセットされ、命令Ａ，Ｄのｐ－ビットはゼロである。このようになっていると、図１３に示す命令順になる。すなわち、命令Ａが実行され、それに続いて命令Ｂ，Ｃ，Ｄが並列に実行される。

【００２２】図システム１４の命令パケットにおいては、命令Ａのｐ－ビットだけが論理１にセットされているので、図１５の実行順序になる。すなわち、命令Ａ，Ｂが並列で実行され、その後に続いて命令Ｃが実行され、最後に命令Ｄが実行される。

【００２３】図１６の命令パケットにおいては、命令Ａ，Ｃのｐ－ビットが１にセットされ、命令Ｂ，Ｄのｐ－ビットはゼロになっているので、図１７に示す実行順序になる。すなわち、命令Ａ，Ｂが並列に実行され、それに続いて命令Ｃ，Ｄが並列に実行される。

【００２４】図１８の命令パケットにおいては、命令Ａ，Ｂのｐ－ビットが１にセットされ、命令Ｃ，Ｄのｐ－ビットはゼロである。このようになっていると、図１９に示す実行順序になる。すなわち、命令Ａ，Ｂ，Ｃが並列に実行され、その後に続いて命令Ｄが実行される。

【００２５】開示した例の中の命令パケットには、４個のプログラム命令が含まれているので、プログラムコンパイラは常に命令Ｄ（４番目の命令）に、０のｐ－ビットを与えてもよい。コンパイラは、命令Ａ，Ｂを並列で実行する規則（propriety ）、命令Ｂ，Ｃを並列で実行する規則、命令Ａ，Ｂ，Ｃを並列で実行する規則に基づいて、命令Ａ，Ｂ，Ｃの残りのｐ－ビットの値を決定する。たとえば、命令Ｂの実行に、命令Ａの実行結果が必要な場合、コンパイラは、命令Ｂが命令Ａの後に実行されるように、命令Ａに０のｐ－ビットを与えるであろ。別の例として、命令Ｂ，Ｃが同一レジスタをアクセスする場合、コンパイラは、命令Ｂ，Ｃが並列ではなく、逐次的に実行されることを保証するため、命令Ｂに０のｐ－ビットを与えるであろう。

【００２６】図２０は、本発明の一実施例を使用するマイクロプロセッサ１のブロック図である。マイクロプロセッサ１は、ＶＬＩＷディジタル信号プロセッサ（”ＤＳＰ”）である。明瞭にするため、図２０は、マイクロプロセッサ１の、本発明の実施例を理解するために関連する部分だけを示している。ＤＳＰの一般的な構造の詳細は公知であり、どこかほかでも容易に調べ出すことができるであろう。たとえば、Frederick Boutaud ほかに発行された米国特許第5,072,418 号は、ＤＳＰを詳細に説明している。ここでこの特許に言及することにより、この特許の開示内容を本願に組み入れることにする。Gary Swobodaほかに発行された米国特許第5,329,471 号は、ＤＳＰのテストとエミュレーションの方法を詳細に説明している。ここでこの特許に言及することにより、この特許の開示内容を本願に組み入れることにする。マイクロプロセッサ技術に関する当業者が本発明を評価して使用できるようにするため、マイクロプロセッサ１の、本発明に関連する部分を以下に十分詳細に説明する。

【００２７】マイクロプロセッサ１には、中央処理装置（ＣＰＵ）１０、データメモリ２２、プログラムメモリ

２３、周辺回路６０およびダイレクトメモリアクセス（ＤＭＡ）を備えた外部メモリインタフェース（ＥＭＩＦ）６１がある。さらにＣＰＵ１０には、命令取り出し／復号化ユニット１０ａ～１０ｃ、演算およびロード／ストアユニットＤ１、乗算器Ｍ１、ＡＬＵ／シフターユニットＳ１、演算論理ユニット（"ＡＬＵ"）Ｌ１、データを読み出しかつデータを書き込む共用マルチポートレジスタファイル２０ａを含む複数の実行ユニットがある。復号化された命令は、示されていないいろいろな制御ラインの組を介して、命令取り出し／復号化ユニット１０ａ～１０ｃから、機能ユニットＤ１，Ｍ１，Ｓ１，Ｌ１に与えられる。データは、第１組のバス３２ａを介してロード／ストアユニットＤ１と、第２組のバス３４ａを介して乗算器Ｍ１と、第３組のバス３６ａを介してＡＬＵ／シフターユニットＳ１と、第４組のバス３８ａを介してＡＬＵＬ１と、レジスタファイル２０ａとの間で送受される。データは、第５組のバス４０ａを介して、メモリ２２とロード／ストアユニットＤ１との間で送受される。上に説明した全データ経路は、レジスタファイル２０ｂおよび実行ユニットＤ２，Ｍ２，Ｓ２，Ｌ２と２重化されていることに注意されたい。命令は、１組のバス４１を介して、取り出しユニット１０ａによって、命令メモリ２３から取り出される。外部のテストシステム５１によって制御されうるエミュレーションユニット５０は、集積回路の内部動作にアクセスできるようにする。

【００２８】メモリ２２，２３は、マイクロプロセッサ１の集積回路の一部として図２０に示されており、その範囲はボックス４２で示されていることに注意されたい。メモリ２２，２３は、マイクロプロセッサ１の集積回路４２の外部にあってもよいし、あるいはメモリの一部が集積回路４２の内部にあり、そしてメモリの一部が集積回路４２の外部にあってもよい。また別の数の実行ユニットを使用してもよい。

【００２９】マイクロプロセッサ１がデータ処理システムに組み込まれると、図１に示すように、メモリまたは周辺装置を追加してマイクロプロセッサ１に接続することができる。たとえば、外部バス７３を介して、ランダムアクセスメモリ（ＲＡＭ）７０、読み出し専用メモリ（ＲＯＭ）７１およびディスク７２が接続されていることが示されている。バス７３は、マイクロプロセッサ４２の内部の機能ブロック６１の重要な部分である外部メモリインタフェース（ＥＭＩＦ）に接続されている。ダイレクトメモリアス（ＤＭＡ）コントローラもブロック６１に含まれている。ＤＭＡコントローラは、一般にマイクロプロセッサ１の内部のメモリと周辺回路との間でデータを転送するために使用されるとともにメモリと、マイクロプロセッサ１の外部にある周辺装置との間でデータを転送するために使用される。

【００３０】本発明の側面から利点を得ることができるいくつかの代表的なシステムは、その開示内容を本願に組み入れた米国特許第5,072,418 号に、特にこの特許の図２から図１８を参照して説明されている。米国特許第5,072,418 号に説明されているシステムをさらに改善するため、性能を改善しかつ原価を低減するため本発明の側面を組み入れたマイクロプロセッサを使用してもよい。そのようなシステムには、工業におけるプロセス制御、自動車（automotive vehicle）システム、モータ制御、ロボット制御システム、衛星通信システム、反響消去システム、モデム、ビデオ映像システム、音声認識システム、暗号付きボコーダーモデムシステムなどが含まれるが、それらに限定されるものではない。

【００３１】図２０のマイクロプロセッサの各種アーキテクチャの特徴は、共同譲渡された特許出願シリアル番号60/036,482（ＴＩ控え番号T-25311）に説明されている。図２０のマイクロプロセッサの命令の完全な命令セットも、共同譲渡された特許出願シリアル番号60/086,482（ＴＩ控え番号T-25311）に説明されている。

【００３２】図２１は、図２０のマイクロプロセッサの実行ユニットおよびレジスタファイルのブロック図であり、各種機能ユニットを接続するバスの詳細な図を示している。本図において、特に注意しない限り、全データバスは３２ビット幅である。バス４０ａには、マルチプレクサ２００ａによって駆動されるアドレスバスＤＡ１がある。このため、ロード／ストアユニットＤ１またはＤ２によって発生したアドレスを、レジスタファイル２０ａに対するロードまたはストアのアドレスにすることができる。データバスＬＤ１は、アドレスバスＤＡ１によって指定されたメモリのアドレスからのデータを、ロードユニットＤ１のレジスタにロードする。ユニットＤ１は、与えられたデータをレジスタファイル２０ａにストアする前に操作（manipulate）する。同様にデータバスＳＴ１は、レジスタファイル２０ａからのデータをメモリにストアする。ロード／ストアユニットＤ１は次の演算、すなわち、３２ビットの加算、３２ビットの減算、線形および循環アドレス計算（linear and circular address calculations）を実行する。ロード／ストアユニットＤ２は、アドレスを選択するマルチプレクサ２００ｂの支援により、Ｄ１と同様に動作する。

【００３３】ＡＬＵユニットＬ１は次のタイプの演算、すなわち、３２／４０ビット数値演算および比較演算、３２ビットの最左端の１ビットおよび０ビットの計数（leftmost 1, 0, bit counting for 32 bits）、３２ビットおよび４０ビットの正規化回数の計数および論理演算を実行する。ＡＬＵ　Ｌ１には、３２ビットのソースオペランドに対する入力src1と、第２の３２ビットのソースオペランドに対する入力src2とがある。入力msb_src は、４０ビットのソースオペランドを生成するために使用される８ビットの値である。ＡＬＵ　Ｌ１には、32ビットの宛先オペランドに対する出力dst がある。出

力msb_dst は、４０ビット宛先オペランドを形成するために使用される８ビットの値である。レジスタファイル２０ａの中にある２つの３２ビットレジスタは、４０ビットのオペランドを保持するために連結されている。マルチプレクサ２１１は入力src1に接続されており、３２ビットのオペランドが、バス３８ａを介してレジスタファイル２０ａから、またはバス２１０を介してレジスタファイル２０ｂから取得されることを可能にしている。マルチプレクサ２１２は入力src2に接続されており、３２ビットオペランドが、バス３８ａを介してレジスタファイル２０ａから取得されること、またはバス２１０を介してレジスタファイル２０ｂから取得されることを可能にしている。ＡＬＵユニットＬ２はユニットＬ１と同様に動作する。

【００３４】ＡＬＵ／シフターユニットＳ１は次のタイプの演算、すなわち、３２ビット数値演算、３２／４０ビットシフトおよび３２ビットビットフィールド演算、３２ビット論理演算、分岐および定数の発生を実行する。ＡＬＵ　Ｓ１には、３２ビットのソースオペランドに対する入力src1と、第２の３２ビットのソースオペランドに対する入力src2とがある。入力msb_src は、４０ビットのソースオペランドを形成するために使用される８ビットの値である。ＡＬＵ　Ｌ１には、４０ビットの宛先オペランドに対する出力ｄｓｔがある。出力msb_dst は、４０ビットの宛先オペランドを生成するために使用される８ビットの値である。マルチプレクサ２１３は、入力src2に接続されており、３２ビットのオペランドが、バス３６ａを介してレジスタファイル２０ａから取得されること、またはバス２１０を介してレジスタファイル２０ｂから取得されることを可能にしている。ＡＬＵＳ２は、ユニットＳ１と同様に動作するが、これに加えて制御レジスタファイル１０２との間の両方向でレジスタ転送を実行する。

【００３５】乗算器Ｍ１は１６×１６の乗算を実行する。ＡＬＵ　Ｓ１には、３２ビット宛先オペランドに対する出力dst がある。乗算器Ｍ１には、３２ビットのソースオペランドに対する入力src1と、３２ビットの宛先オペランドに対する入力src2とがある。マルチプレクサ２１４は入力ｓｒｃ２に接続されており、３２ビットオペランドが、バス３４ａを介してレジスタファイル２０ａから取得されること、またはバス２１０を介してレジスタファイル２０ｂから取得されることを可能にしている。乗算器Ｍ２は、乗算器Ｍ１と同様に動作する。

【００３６】図２２は、図１のマイクロプロセッサにおける命令実行パイプラインの処理フェーズを示す図表である。各フェーズは、システムクロックのクロックサイクルにほぼ対応している。たとえば、マイクロプロセッサ１が２００メガヘルツで動作している場合、各フェーズは通常５ナノ秒である。しかし、ＲＡＭ７０のようなメモリあるいは周辺装置からデータが期待されている場合、期待されているときにデータが用意されていないとパイプラインはストール(stall）する。パイプラインがストールすると、いくつかのシステムクロックサイクルの間、あるパイプラインフェーズになっている。

【００３７】図２２において、命令を処理する第１フェーズは、フェーズＰＧでプログラムアドレスを発生させることである。これは、制御レジスタファイル１０２に配置されているプログラム取り出しカウンタＰＦＣをロードィングすることにより実行される。第２の命令処理フェーズＰＳ中に、バス４１の役割の１つであるプログラムアドレスバスＰＡＤＤＲを介して、命令取り出しパケットのアドレスがプログラムメモリ２３に送られる。第３のフェーズＰＷは、メモリ２３におけるアクセス時間を考慮するための待ち合わせフェーズである。第４のフェーズＰＲ中に、バス４１の一部であるデータバスＰＤＡＴＡ＿Ｉを介してプログラムメモリ２３からプログラム取り出しパケットを使用することができる。第５の処理フェーズＤＰ中に命令の並列性が検出されると、実行できる命令が適切な機能ブロックにディスパッチされる。パイプライン動作のこの側面については、後の節でさらに詳細に説明する。第６の処理フェーズＤＣ中に、実行可能な命令が復号化され、制御信号が発生して、各種データ経路と機能ユニットとを制御する。

【００３８】図２３は、図１のマイクロプロセッサ１における命令実行パイプラインの実行フェーズを示す図表である。第１の実行フェーズＥ１中に、″ＩＳＣ″と呼ぶ１サイクル命令と、″ＢＲ″と呼ぶ分岐命令が終了する。指定された実行ユニットは、制御回路１００によって指示されるとおり、図２３に示す演算を実行する。第２の実行フェーズＥ２中に、制御回路１００の制御によって指定された実行ユニットにより、次の命令、すなわち、整数の乗算（ＩＭＰＹ）、プログラムストア命令（ＳＴＰ）およびデータストア命令（ＳＴＤ）を終了させる。第３の実行フェーズＥ３中に、データメモリシステム（ＤＭＳ）からのデータをラッチすることにより、指示されたとおり、ロードデータ命令（ＬＤ）の実行を継続する。第４の実行フェーズＥ４中に、実行ユニットＤ１または実行ユニットＤ２のデータ入力レジスタＤＤＡＴＡ＿Ｉに、Ｅ３でラッチされたデータが転送される。第５の実行フェーズＥ５中に、レジスタＤＤＡＴＡ＿Ｉのデータを操作して、その操作したデータを、レジスタファイル２０ａまたは２０ｂの中の指定されたレジスタに書き込むことにより、ＬＤ命令が終了する。

【００３９】図２４は、図２２の処理フェーズ中に命令取り出しパケットを処理するタイミングの詳細と、図２３の実行フェーズ中の実行パケットの実行とを示すタイミング図である。フェーズＰＳではプログラムメモリレディ信号ＰＲＤＹがローのため、フェーズＰＷにパイプストールが示され、フェーズＥ２ではデータメモリレディ信号ＤＲＤＹがローのため、フェーズＥ３に第２のパ

イプストールが示されていることに注意されたい。
【００４０】図２５は、図２０のマイクロプロセッサにおける命令のディスパッチ動作を示すブロック図である。本実施例において、命令取り出しパケットは８つの命令を含んでいる。命令取り出しパケット１７１０は、図示の通り８つの実行ユニットにディスパッチされて復号化される。取り出しパケット１７２０は分岐命令１７２５を含んでいる。取り出しパケット１７３０は３つの命令実行パケットを含んでいる。第１の実行パケットは、２つの命令、ＺＥＲＯ、ＳＨＬを含んでおり、これらの命令は、分岐命令１７２５に関する第１遅延スロットにおける処理を開始する。第２の実行パケットは、４つの命令、ＡＤＤ，ＳＩＢ，ＳＴＷ，ＳＴＷを含んでおり、これらの命令は、分岐命令１７２５に関する第２遅延スロットにおける処理を開始する。第３の実行パケットは、２つの命令、ＡＤＤＫ，ＢＲを含んでおり、これらの命令は、分岐命令１７２５の第３遅延スロットの処理を開始する。
【００４１】並列演算
命令は常に１度に８つ取り出される。これが１つの取り出しパケットを構成する。取り出しパケットの基本フォーマットは、図２６に示されている。取り出しパケットの実行のグループ化は、各命令のビット０のｐ－ ビットによって指定される。取り出しパケットは８語に揃えられている。
【００４２】ｐ－ ビットは、命令の並列実行を制御する。ｐ－ ビットは左から右へ（低いアドレスから高いアドレスへ）走査される。命令ｉのｐ－ ビットが１の場合、命令ｉ＋１は命令ｉと（命令ｉと同じサイクルで）並列に実行されることになる。命令ｉのｐ－ ビットが０の場合、命令ｉの後のサイクルで命令ｉ＋１が実行される。並列で実行する全命令は、１つの実行パケットを構成する。１つの実行パケットは最大８つの命令を含むことができる。実行パケットの中の全命令は独自の機能ユニットを使用なければならない。
【００４３】実行パケットは、８語の境界を横切ることはできない。したがって、取り出しパケットの最後のｐ－ ビットは常に０にセットされており、各取り出しパケットは、新しい実行パケットを開始させる。図４から図１９について考察したとおり、取り出しパケットに対して３タイプのｐ－ ビットのパターンがある。これらの３つのｐ－ ビットパターンは、次に示す８つの命令の実行順序になる。すなわち、完全シリアル、完全並列、部分的シリアルの３つである。

```
        ADD  S1   A0,A1,A2  ;＼
    ||  SHR .S1   A3,15,A4  ;／
```

【００４４】並列符号の例
文字″ ||″ は、命令が直前の命令と並列に実行されることを表している。図２７の取り出しパケットにおいて、この符号は次のように表示される。
【表１】

```
        命令A

        命令B

        命令C
    ||  命令D
    ||  命令E

        命令F
    ||  命令G
    ||  命令H
```

【００４５】実行パケットの中間に対する分岐動作
実行パケットの中間に対する分岐が発生すると、それよりも低いアドレスの全命令は無視される。図２７の例において、命令Ｄを含むアドレスに対する分岐が発生すると、ＤおよびＥだけが実行される。命令Ｃも同一実行パケットの中にあるが、命令Ｃは無視される。また命令Ａ，Ｂは時間的に前の方の実行パケットにあるから、命令Ａ，Ｂも無視される。
【００４６】資源の制約
同一実行パケットの中のどの２つの命令も同一資源を使用することはできない。また、どの２つの命令も、同一サイクル中に同一レジスタに書き込むことはできない。以下の節において、命令が使用できる資源のそれぞれについて説明する。
【００４７】機能ユニット
同一機能ユニットを使用する２つの命令を同一実行パケットで発行することはできない。
【００４８】次の実行パケットは無効である。
【表２】

```
        ADD  L1   A0,A1,A2  ;＼  .S1が両命令に使用されている。
    ||  SHR .S1   A3,15,A4  ;／
```

【００４９】次の実行パケットは有効である。

```
        ADD  L1   A0,A1,A2  ;＼
    ||  SHR .S1   A3,15,A4  ;／
```

【表３】
２つの異なる機能ユニットが使用されている。
【００５０】相互経路(Cross Paths)（１Ｘおよび２Ｘ）
実行ユニットごとの（Ｓ，Ｌ，Ｍのいずれか）１つのユニットは、データ経路ごとに相互経路（１Ｘおよび２Ｘ）を介してその反対側のレジスタファイルからソースオペランドを読み出すことができる。たとえば、Ｓ１は、Ａレジスタファイルから両方のオペランドを読み出すことができるか、あるいは１Ｘ相互経路を使用して、Ｂレジスタファイルから１つのオペランドを読み出すことができる。このことは、ユニット名の後にくるＸによ

って表示される。

【００５１】ＡからＢの間およびＢからＡの間には、それぞれ１つの経路しかないのであるから、レジスタファイル間で同一Ｘ相互経路を使用する２つの命令を同一実

```
ADD.L1X   A0,B1,A1   ;＼
||  MPY.M1X   A4,B4,A5   ;／
```

【００５３】次の実行パケットは有効である。

```
ADD.L1X   A0,B1,A1   ;＼
||  MPY.M2X   A4,B4,132  ;／
```

【００５４】命令フィールドのｘビットがセットされている場合、オペランドは宛先と反対のレジスタファイルから来る。

【００５５】ロード経路およびストア経路

ロードおよびストアは、他のレジスタファイルへのロードまたは他のレジスタファイルからのストアを実行しながら、１つのレジスタファイルからのアドレスポインタ

```
LDW D1    *A0,A1     ;＼
||  LDW.D1    *A2,B2     ;／
```

次の実行パケットは有効である。

```
LDW D1    *A0,A1     ;＼
||  LDW.D2    *B0,B2     ;／
```

【００５７】同一レジスタファイルにロード中の、および／あるいは同一レジスタファイルからストア中の２つのロードおよびストアを、同一実行パケットで発行する

```
LDW D1    *A4,A5     ;＼
||  STW.D2    A6,*B4     ;／
```

【００５９】次の実行パケットは有効である。

```
LDW D1    *A4,B5     ;＼
||  STW.D2    A6,*B4     ;／
```

【００６０】長い経路(Long Path)

１サイクル当たり、１つだけの長い結果(long result)を両側のレジスタファイルに書き込むことができる。Ｓユニットおよび．Ｌユニットは、長いソースオペランドの読み出しレジスタポートと、長い結果の書き込みレジ

```
ADD.L1    A5:A4,A1,A3:A2   ;＼
||  SHL.S1    A8,A9,A7:A6      ;／
```

次の実行パケットは有効である。

```
ADD.L1    A5:A4,A1,A3:A2   ;＼
||  SHL.S2    B8,B9,B7:B6      ;／
```

【００６２】．Ｌユニットおよび．Ｓユニットは、ストアポートと、これら両ユニット用の長い読み出しレジスタを共用しているので、ストアと同じ実行パケットで、長い値を読み出す演算を．Ｌユニットおよび／あるい

```
ADD.L1    A5:A4,A1,A3:A2   ;＼
||  STW.D1    A8, *A9          ;／
```

次の実行パケットは有効である。

```
ADD.L1    A4,A1,A3:A2      ;＼
||  STW.D1    AB, *A9          ;／
```

【００６４】レジスタ読み出し

同一サイクルで、同一レジスタに対する４回以上の読み

行パケットで発行することはできない。

【００５２】次の実行パケットは無効である。

【表４】

```
両命令に対して１Ｘ相互経路が
使用されている。
```

【表５】

```
これらの命令は、１Ｘおよび２Ｘ
相互経路を使用している。
```

を使用することができる。同一レジスタファイルからのアドレスポインタを使用する２つのロードおよび／あるいはストアを同一実行パケットで発行することはできない。

【００５６】次の実行パケットは無効である。

【表６】

```
同一レジスタファイルからの
アドレスレジスタである。
```

【表７】

```
異なるレジスタファイルからの
アドレスレジスタである。
```

ことはできない。

【００５８】次の実行パケットは無効である。

【表８】

```
同一レジスタファイルとのロード
およびストアである。
```

【表９】

```
異なるレジスタファイルとのロード
およびストアである。
```

スタポートとを共用しているので、実行パケットで、１側に１つだけこの種命令を発行してもよい。

【００６１】次の実行パケットは無効である。

【表１０】

```
;＼ Ａレジスタファイルへの
;／ 長い書き込みが２つある。
```

【表１１】

```
;＼ 各レジスタファイルへの
;／ 長い書き込みが１つある。
```

は．Ｓユニットに発行することはできない。

【００６３】次の実行パケットは無効である。

【表１２】

```
;＼ 長い読み出し演算
;／ とストアがある。
```

【表１３】

```
;＼ ストアと一緒の長い
;／ 読み出しがない。
```

出しを発生させることはできない。条件付きレジスタはこの回数に含まれていない。

【００６５】次のコードシーケンスは無効である。

```
        MPY     .M1   A1,A1,A4    ;  レジスタＡ１を５回
||      ADD     .L1   A1,A1,A5       読出している。
||      SUB     .D1   A1,A2,A3
```

【００６６】一方このコードシーケンスは有効である。

```
                MPY     .M1   A1,A1,A4    ;  レジスタＡ１を４回
||      [A1]    ADD     .L1   A0,A1,A5       だけ読出している。
                SUB     .D1   A1,A2,A3
```

【００６７】レジスタ書き込み

同一レジスタに書き込む命令であって、異なる待ち合わせ時間をもつ命令が、異なるサイクルに発行される場合、同一サイクルで、同一レジスタに複数回の書き込みが発生することがある。たとえば、サイクルｉ＋１のＡＤＤの前のサイクルｉで発行されたＭＰＹは、同一レジスタに書き込みをすることができない。その理由は、両命令がサイクルｉ＋１で結果を書き込むからである。したがって、次のコードシーケンスは無効である。

【表１６】

```
L1:             ADD.L2    B5,B6,B7   ;  検出可能、競合する
||              SUB.S2    B8:B9,B7
L2:             MPY.M2    B0,B1,B2   ;＼ 検出不可能
L3:             ADD.L2    B3,B4,B2   ;／
L4:     [!B1]   ADD.L2    B5,B6,B7   ;  検出可能、競合しない
||      [B0]    SUB.S2    B8,B9,B7
L5:     [!B1]   ADD.L2    B5,B6,B7   ;＼ 検出不可能
||      [B0]    SUB.S2    B8,B9,B7   ;／
```

【００６９】パケットＬ２のＭＰＹとパケットＬ３のＡＤＤは、共に同時にＢ２に書き込むが、分岐命令のため、Ｌ２の後の実行パケットが、Ｌ３以外の何らかのパケットになると、これは競合にならないであろう。したがって、Ｌ２およびＬ３における潜在的な競合は、アセンブラによって検出されないかもしれない。Ｌ４の中の命令は書き込み競合を構成しない。何故かというと、これらの命令は相互に排他的だからである。対照的に、Ｌ５の中の命令は相互に排他的であることが明確でないから、アセンブラは競合を決定することができない。同一レジスタに対して書き込みを複数回実行するコマンドをパイプラインが受信すると、その結果は定義されていない。

【００７０】本発明の模範的な例が上に説明されているが、この実施例は本発明を限定するものではない。本発明は、多様な実施例の中で実現されうる。

【００７１】関連特許に対する相互参照

本願は、本願と同時に出願され、共通譲渡された特許出願シリアル番号第60/036,482号（ＴＩ控え番号ＴＩ－２１３５）に関連している。ここでこの特許出願に言及することにより、この特許出願の開示内容を本願に組み入れることにする。

【図面の簡単な説明】

【図１】本発明によるデータ処理システムのブロック図。

【表１４】

```
        MPY     .M1   A1,A1,A4    ;  レジスタＡ１を５回
||      ADD     .L1   A1,A1,A5       読出している。
||      SUB     .D1   A1,A2,A3
```

【表１５】

```
                MPY     .M1   A1,A1,A4    ;  レジスタＡ１を４回
||      [A1]    ADD     .L1   A0,A1,A5       だけ読出している。
                SUB     .D1   A1,A2,A3
```

```
        MPY     .M1   A0,A1,A2
        ADD     .L1   A4,A5,A2
```

【００６８】書き込み競合（conflict）の検出可能性

次の実行パケットのシーケンスは、異なる複数の書き込みの競合を示している。たとえば、実行パケットＬ１の中のＡＤＤおよびＳＵＢは、同一レジスタに書き込む。この競合は容易に検出できる。

【表１７】

【図２】図１のデータ処理回路の一部のブロック図。

【図３】本発明で使用されている命令パケットの基本フォーマットを示す図。

【図４】図３のフォーマットによる命令パケットの一例を示す図。

【図５】図４の命令パケットによって定義される実行順序を示す図。

【図６】図３のフォーマットによる命令パケットの別の例を示す図。

【図７】図６の命令パケットによって定義される実行順序を示す図。

【図８】図３のフォーマットによる命令パケットの別の例を示す図。

【図９】図８の命令パケットによって定義される実行順序を示す図。

【図１０】図３のフォーマットによる命令パケットの別の例を示す図。

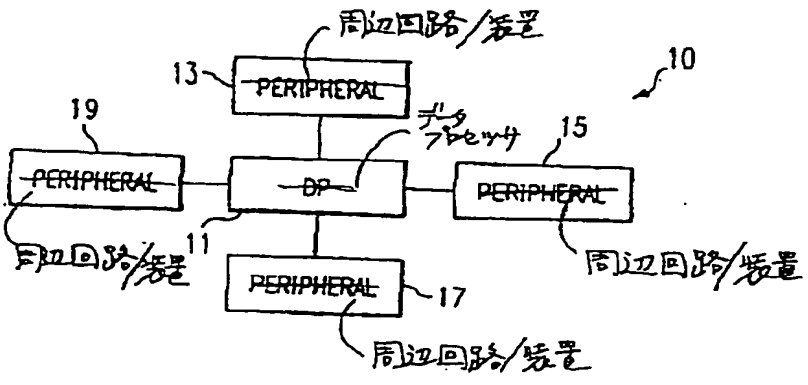
【図１１】図１０の命令パケットによって定義される実行順序を示す図。

【図１２】図３のフォーマットによる命令パケットの別の例を示す図。

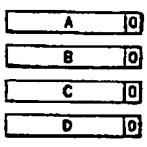
【図１３】図１２の命令パケットによって定義される実行順序を示す図。

【図１４】図３のフォーマットによる命令パケットの別の例を示す図。
【図１５】図１４の命令パケットによって定義される実行順序を示す図。
【図１６】図３のフォーマットによる命令パケットの別の例を示す図。
【図１７】図１６の命令パケットによって定義される実行順序を示す図。
【図１８】図３のフォーマットによる命令パケットの別の例を示す図。
【図１９】図１８の命令パケットによって定義される実行順序を示す図。
【図２０】本発明の実施例を使用するマイクロプロセッサのブロック図。
【図２１】図２０のマイクロプロセッサの実行ユニットおよびレジスタファイルのブロック図。
【図２２】図２０のマイクロプロセッサにおける命令実行パイプラインの処理フェーズを示す図表。
【図２３】図２０のマイクロプロセッサにおける命令実行パイプラインの処理フェーズを示す図表。
【図２４】図２２の処理フェーズ中に命令取り出しパケットを処理するタイミングと、図２３の実行フェーズ中の実行パケットの実行とを示すタイミング図。
【図２５】図２０のマイクロプロセッサにおける命令のディスパッチを示すブロック図。
【図２６】図２０のマイクロプロセッサの命令取り出しパケットの基本フォーマットを示す図。
【図２７】部分的に並列な８語取り出しパケットを示す図。
【符号の説明】
１　マイクロプロセッサ
１０　データ処理システム／中央処理装置（ＣＰＵ）
１０ａ　命令取り出し
１０ｂ　命令ディスパッチユニット
１０ｃ　復号化ユニット
１１　データ処理回路
１３，１５，１７，１９　周辺回路
１２ａ，１２ｂ　演算およびロード／ストアユニット
１４ａ．１４ｂ　乗算器
１６ａ．１６ｂ　ＡＬＵ／シフターユニット
１８ａ，１８ｂ　数値・論理演算ユニット
２０ａ、２０ｂ　レジスタファイル
２１　プログラムカウンタ
２２　データメモリ
２３，１７２３　プログラムメモリ
２３　取り出し回路
２５　メモリ
２７　命令レジスタ
２９　プログラム実行回路
３２ａ，３４，３６ａ、３８ａ．４０ａ　バス
４１　バス
４２　マイクロプロセッサの範囲を示すボックス
５０　エミュレーションユニット
５１　外部テストシステム
６０　周辺回路
６１　ダイレクトメモリアクセス（ＤＭＡ）付き外部メモリインタフェース
７３　外部バス
７０　ＲＡＭ
７１　ＲＯＭ
７２　ディスク装置
１００　制御論理／制御回路
１０２　制御レジスタファイル
２００ａ，２００ｂ，２１１，２１２，２１３，２１４　マルチプレクサ
１７１０，１７２０，１７３０，１７４０　命令取り出しパケット
１７２５，１７３８　分岐命令

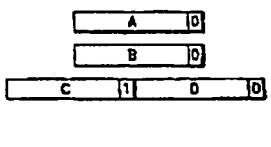
【図１】



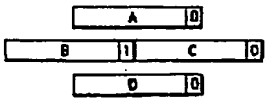
【図５】



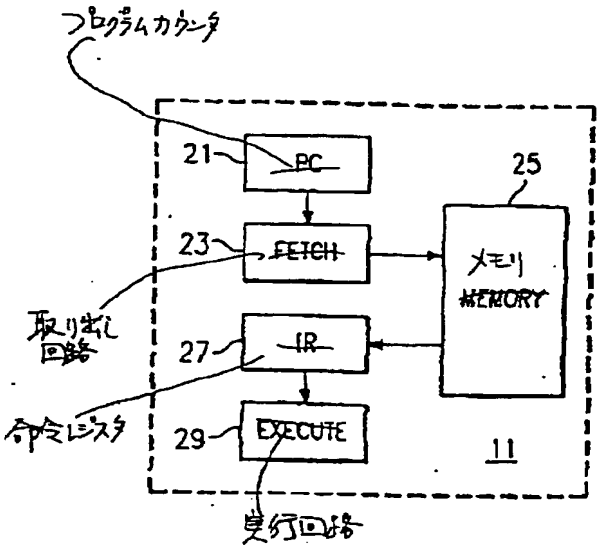
【図９】



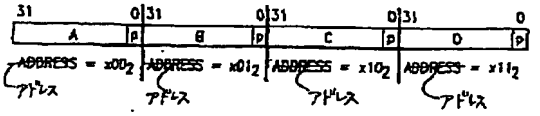
【図１１】



【図２】

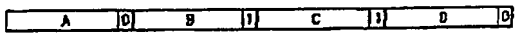


【図３】

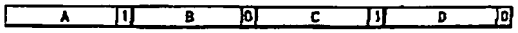


【図７】

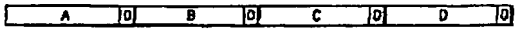
【図１２】



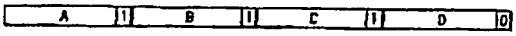
【図１６】



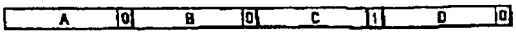
【図４】



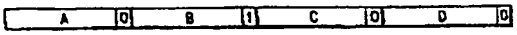
【図６】



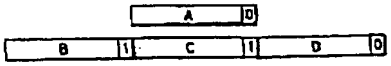
【図８】



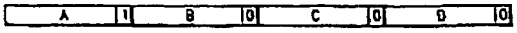
【図１０】



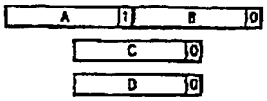
【図１３】



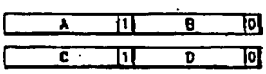
【図１４】



【図１５】

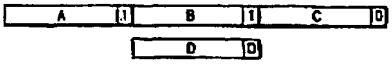


【図１７】

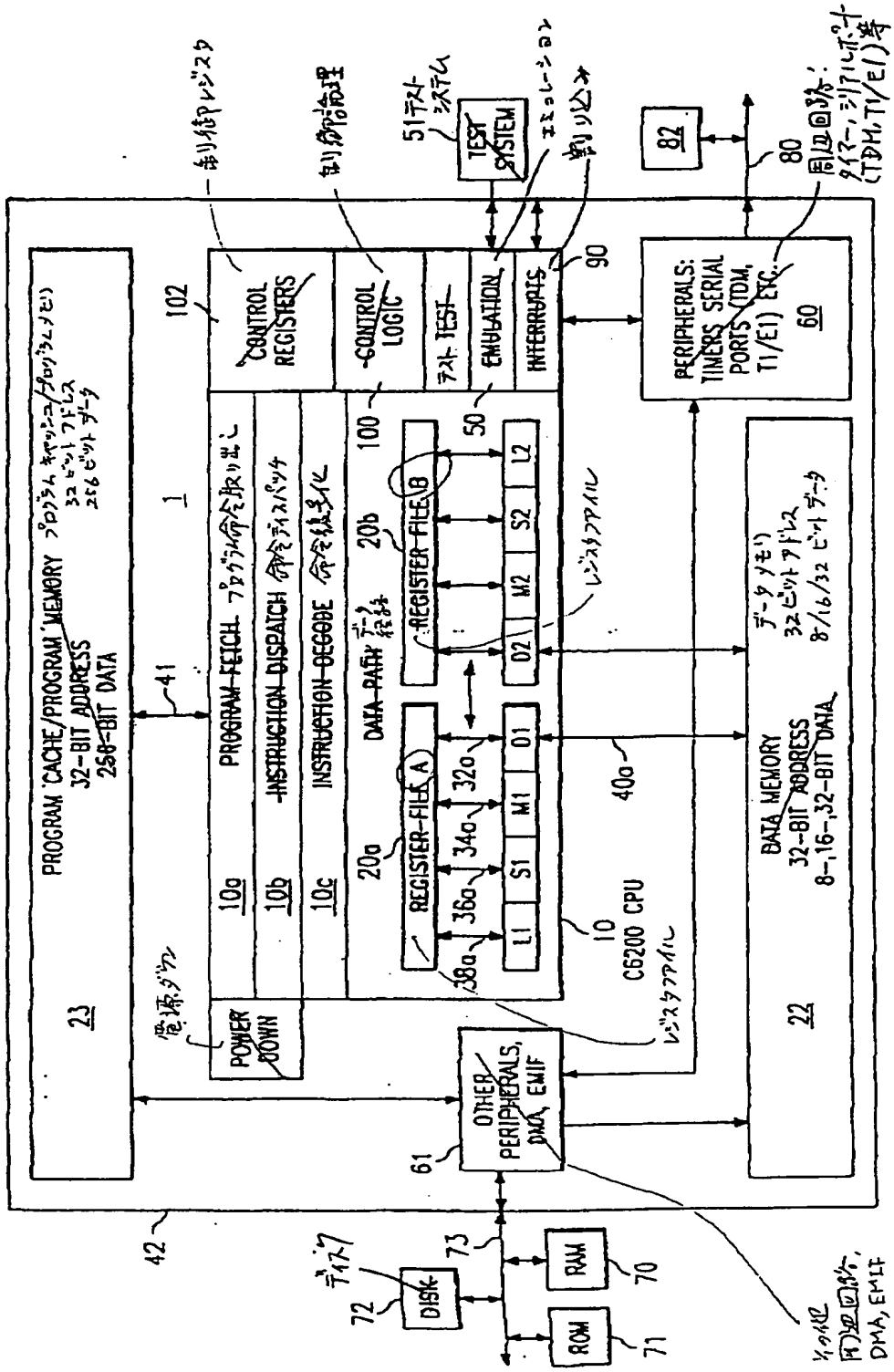


【図１８】

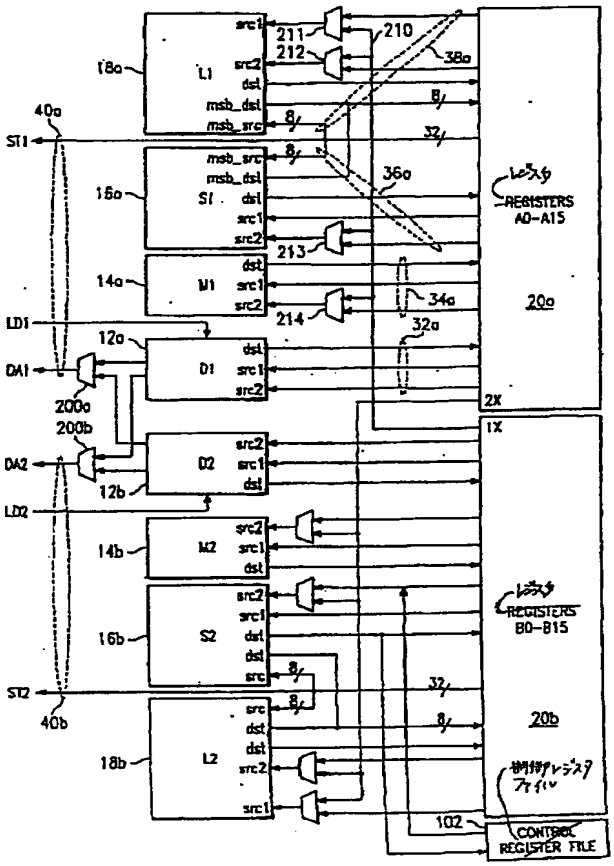
【図１９】



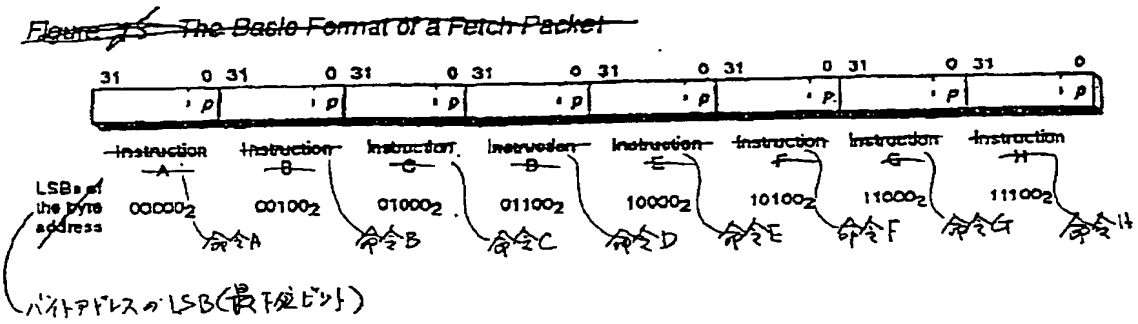
【図20】


PROGRAM CACHE/PROGRAM MEMORY
32-BIT ADDRESS
256-BIT DATA
23
1
102
CONTROL REGISTERS
CONTROL LOGIC
テスト TEST
EMULATION
INTERRUPTS
90
51 テスト システム
TEST SYSTEM
エミュレーション
割り込み
82
80
周辺回路：タイマー、シリアルポート等 (TDM, T1/E1)
PERIPHERALS: TIMERS SERIAL PORTS (TDM, T1/E1) ETC.
60
PROGRAM FETCH
INSTRUCTION DISPATCH
INSTRUCTION DECODE
DATA PATH
10a
10b
10c
20a
20b
100
50
REGISTER FILE B
REGISTER FILE A
L2 S2 M2 D2
L1 S1 M1 D1
32a 34a 36a 38a
レジスタファイル
40a
10
C6200 CPU
POWER DOWN
電源ダウン
OTHER PERIPHERALS, DMA, EMIF
61
22
DATA MEMORY
32-BIT ADDRESS
8-,16-,32-BIT DATA
データメモリ
32ビットアドレス
8/16/32 ビットデータ
42
73
DISK
72
ディスク
RAM
70
ROM
71
外部周辺回路：DMA, EMIF

【図２１】


L1
S1
M1
D1
D2
M2
S2
L2
src1
src2
dst
msb_dst
msb_src
211
212
213
214
210
36a
38a
34a
32a
REGISTERS A0-A15
REGISTERS B0-B15
20a
20b
2X
1X
102
CONTROL REGISTER FILE
LD1
DA1
LD2
DA2
ST1
ST2
40a
40b
200a
200b
12a
12b
14a
14b
16a
16b
18a
18b

【図２６】


Instruction A
Instruction B
Instruction C
Instruction D
Instruction E
Instruction F
Instruction G
Instruction H
LSBs of the byte address
00000₂
00100₂
01000₂
01100₂
10000₂
10100₂
11000₂
11100₂
パケットアドレスのLSB(最下位ビット)

【図22】

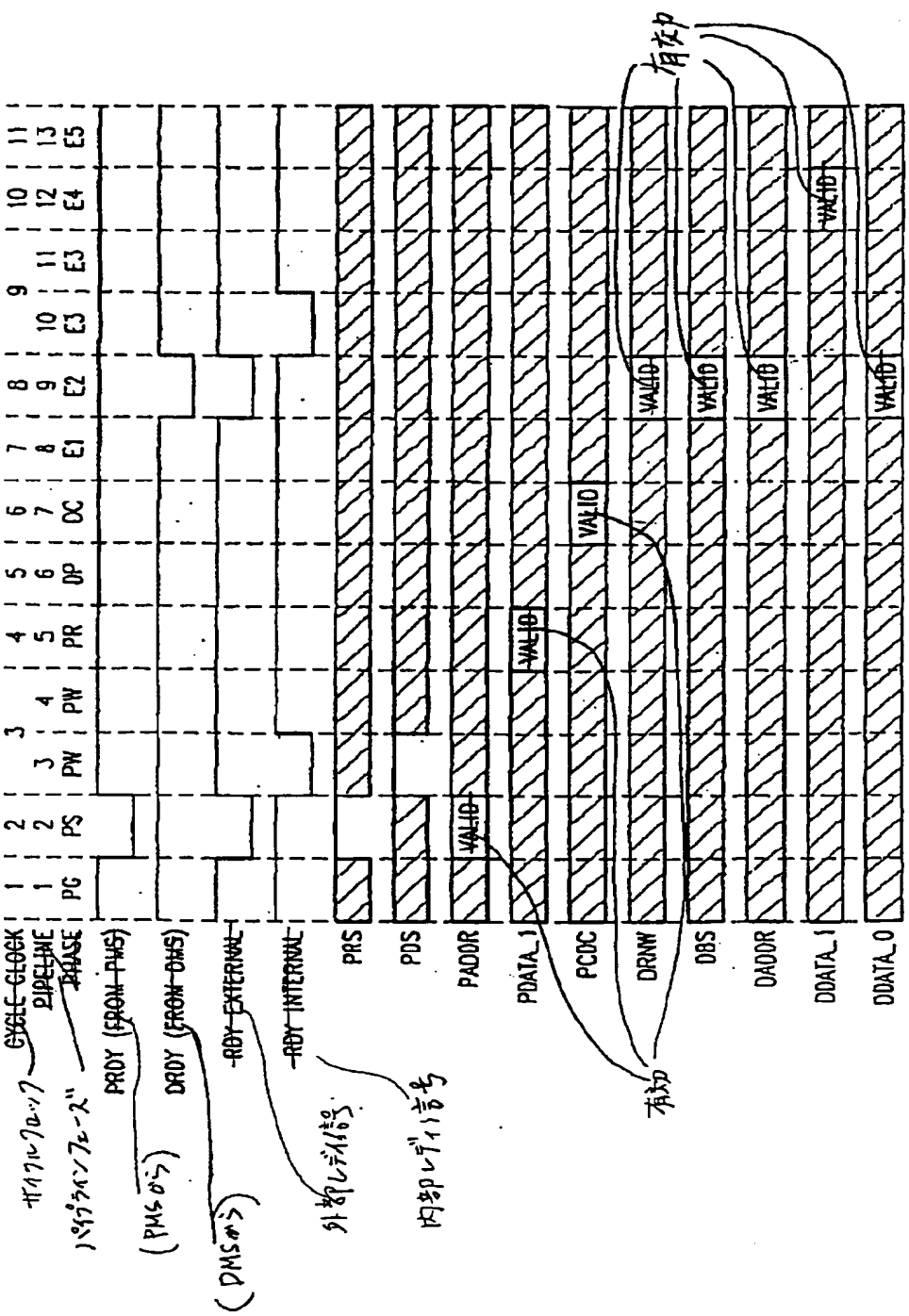
パイプラインフェーズ	~~Symbol~~ 符号	~~During this Entire Phase~~ このフェーズ中に	~~By the End of This Phase~~ このフェーズの終了時点で
プログラムアドレス発生	PG		□プログラムストアがＥ１にある場合を除き、ＰＦＣおよびＰＡＤＤＲは、それらの次の値、すなわち、現在のＰＦＣの後の８語境界か、分岐命令の目標アドレスのいずれかにラッチされる。 □プログラムストアがＥ１にある場合、ＰＦＣは変更されずその値のままであり、ＰＡＤＤＲはプログラムストア命令からの値をとる。
プログラムアドレス送信	PS	□PADDR上でPFCが有効である。 □取り出しパケットを取り出す必要がある場合、ＰＲＳがアクティブにセットされる。	□ＰＲＳがアクティブの場合、ＰＭＳがＰＡＤＤＲをラッチする。
プログラム待ち合わせ	PW	□最終ＰＲＳで取り出しパケットが指示された場合、ＰＤＳが立ち上げられる。	□ＰＤＳが立ち上げられると、ＰＭＳは、ＰＲ中に駆動されるように、最後に要求された取り出しパケットをラッチする。
Program Data Receive プログラムデータ受信	PR	□関連するＰＷ中にＰＤＳがアクティブの場合、ＰＭＳは、要求された取り出しパケットをＰＤＡＴＡ＿１上に駆動する。それ以外の場合、ＰＤＡＴＡ＿１上に駆動された前の取り出しパケットが維持される。	□ＰＷ中のアクティブなＰＤＳによって要求された取り出しパケットが必要なことを決定することができると、ＣＰＵはこの取り出しパケットをＰＤＡＴＡ＿１上にラッチする。それ以外の場合、このラッチは状態を維持する。
~~Dispatch~~ ディスパッチ復号化	DP	PCDCが有効である	□取り出しパケットの並列性が検出される。 □取り出しパケットで次に要求された実行パケットの中の命令が、反対側の機能ユニットにディスパッチされる。 □検出されたＣＲＥＧフィールドの中でソフトウエア中断点コードが検出される。 □命令：複数サイクルＮＯＰ，ＮＯＰ，ＳＷＩ，ＩＤＬＥが、個別の機能ユニットに転送されないため復号化される。 □そのサイクルの終了時点で、ＤＰ中の実行パケットのＰＣが、ＰＣＤＣとしてラッチされる。
~~Decode~~	DC	□ ~~PCDC valid~~	□実行中のアクションの制御信号が復号化される。

【図２３】

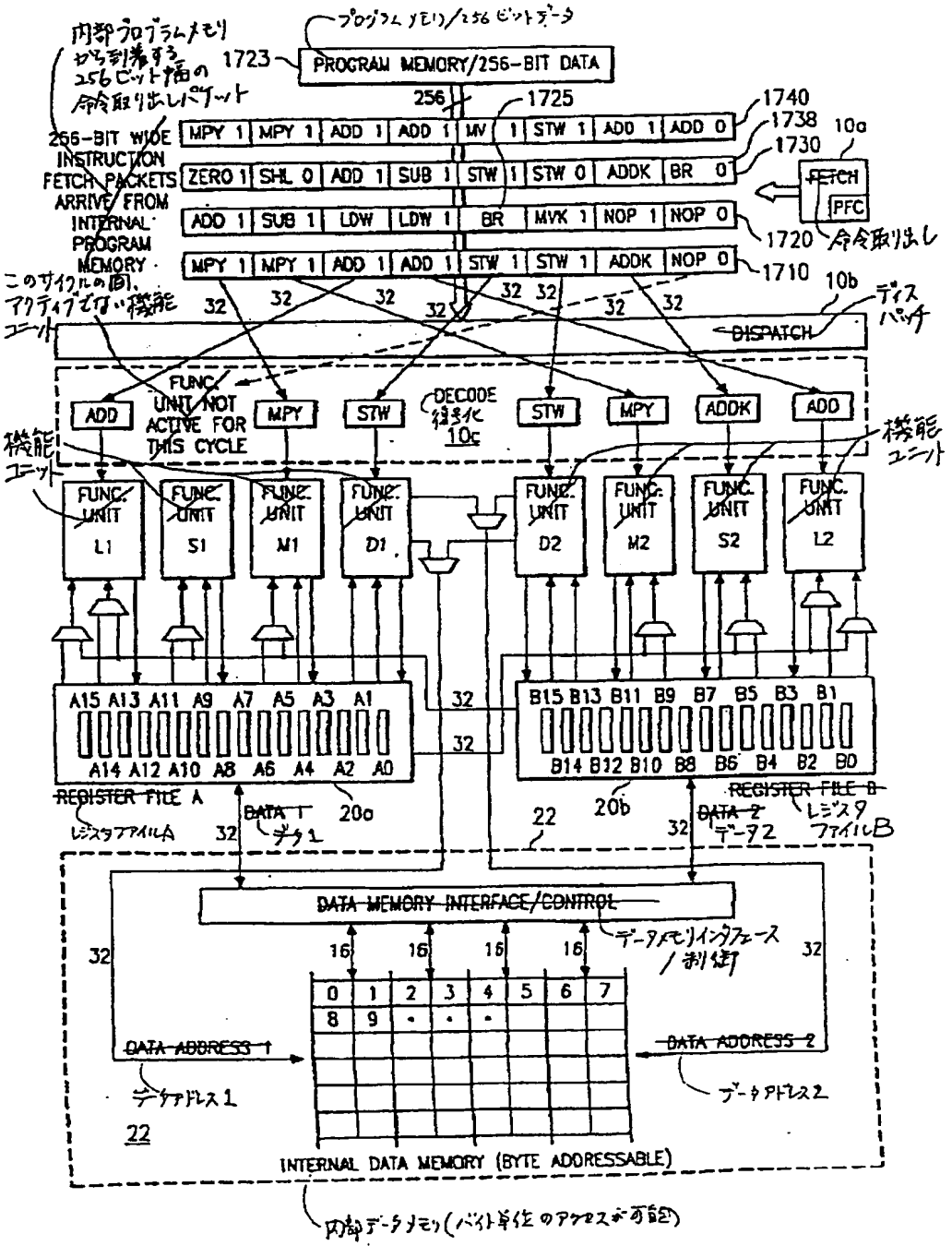
Pipeline Phase パイプラインフェーズ	Symbol 符号	During this Phase このフェーズ中に	By the End of This Phase このフェーズの終了時点で	CPU has Completed Types CPUが終了させた形式
Execute 1 実行	E1	□ All Types: Registers read. 全形式：レジスタが読み出される	□ＩＳＳ形式：結果がレジスタファイルに書き込まれる。 □ＩＳＳ形式：状態が制御レジスタファイルに書き込まれる。 □ＬＤおよびＳＴ形式：アドレス修飾がレジスタファイルに書き込まれる。 □ＬＤおよびＳＴ形式：次のフェーズのためＤＡＤＤＲ、ＤＢＳおよびＤＲＮＷがラッチされる。 □ＳＴ形式：次のフェーズのためＤＤＡＴＡ＿０がラッチされる。 □BRANCH形式：PFCおよびPADDRが目標アドレスをラッチする。 □ＳＴＰ形式：取り出し中でなければ、ＰＡＤＤＲが宛先アドレスをラッチする。次のフェーズのためPWSがアクティブにラッチされる。 □LD,ST,STP,IMPY形式：中間状態がラッチされる。	□ ISC. □ BR.
実行 Execute 2	E2	□LDおよびST形式：DBSがアクティブでDADDRが有効である。 □LD形式：DRNWがアクティブである。 □ST形式：DRNWがインアクティブでDDATA＿０が有効である。	□ＩＭＰＹ形式：結果ががレジスタファイルに書き込まれる。 □ＩＭＰＹ形式：状態が制御レジスタファイルに書き込まれる。 □ＬＤおよびＳＴ形式：ＤＭＳがＡＤＤＲをラッチする。 □ＳＴ形式：ＤＭＳがＤＤＡＴＡ＿０をラッチする。 □STP形式：ＤＭＳがＰＤＡＴＡ＿０をラッチする □ＬＤ形式：中間状態がラッチされる。	□ IMPY. □ STD. □ STP.
Execute 3 実行	E3		□LD形式：DMSは最終サイクル中に要求されたデータをラッチする。 □ＬＤ形式：中間状態がラッチされる。	
Execute 4 実行	E4	□LD形式：DMSはDDATA-1上で関連するE2中に要求されたデータを駆動する。	□ＬＤ形式：ＤＤＤＡＴ＿１がラッチされる。 □ＬＤ形式：中間状態がラッチされる。	
Execute 5	E5		□LD形式：DDATA＿１が符号を拡張するかゼロを埋め込んで右に揃えられ、レジスタファイルに書き込まれる。	□ LD.

特開平１０－２３２７７９

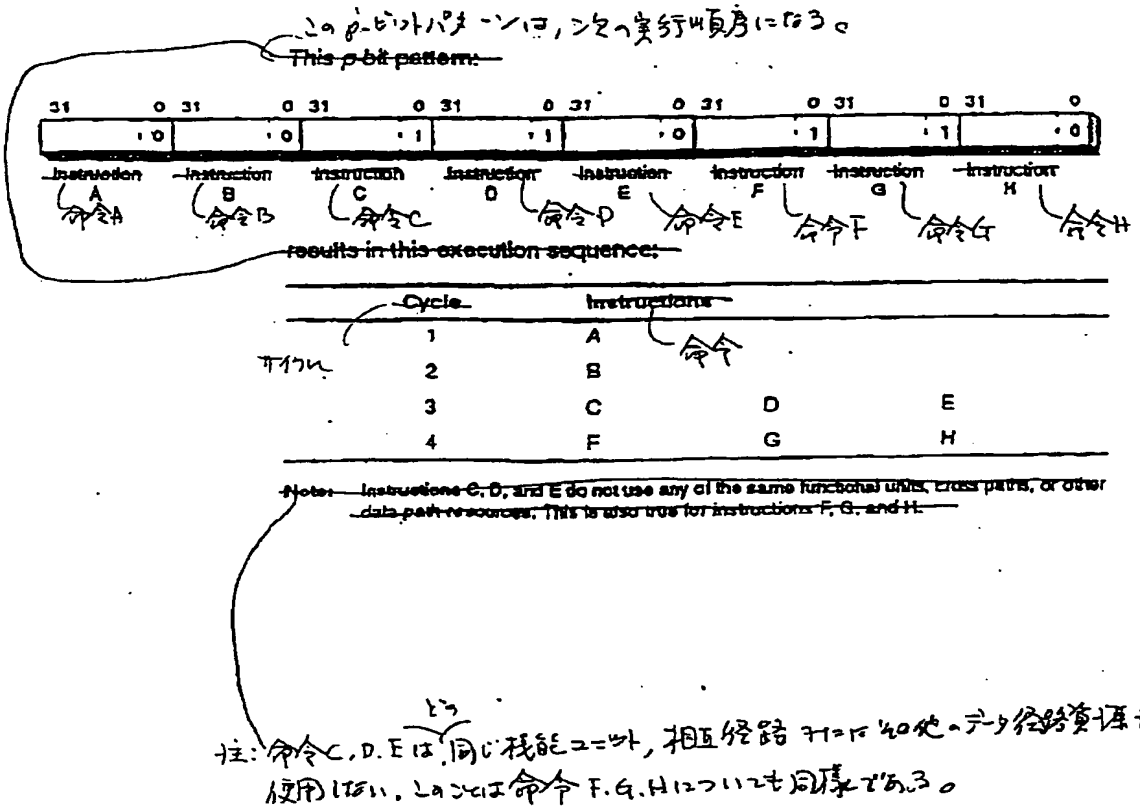
【図２４】


CYCLE CLOCK
PIPELINE PHASE
PRDY (FROM PMS)
DRDY (FROM DMS)
RDY EXTERNAL
RDY INTERNAL
PRS
PDS
PADOR
PDATA_1
PCDC
DRNW
DBS
DADOR
DDATA_1
DDATA_0
VALID
有効
外部レディ信号
内部レディ信号

【図２５】


内部プログラムメモリから到着する256ビット幅の命令取りだしパケット
プログラムメモリ/256ビットデータ
1723
PROGRAM MEMORY/256-BIT DATA
256
1725
1740
1738
1730
10a
256-BIT WIDE INSTRUCTION FETCH PACKETS ARRIVE FROM INTERNAL PROGRAM MEMORY
MPY 1 MPY 1 ADD 1 ADD 1 MV 1 STW 1 ADD 1 ADD 0
ZERO 1 SHL 0 ADD 1 SUB 1 STW 1 STW 0 ADDK BR 0
ADD 1 SUB 1 LDW LDW 1 BR MVK 1 NOP 1 NOP 0
MPY 1 MPY 1 ADD 1 ADD 1 STW 1 STW 1 ADDK NOP 0
FETCH
PFC
1720
命令取りだし
1710
このサイクルの間、アクティブでない機能ユニット
10b
ディスパッチ
DISPATCH
FUNC. UNIT NOT ACTIVE FOR THIS CYCLE
ADD MPY STW DECODE 解読化 10c STW MPY ADDK ADD
機能ユニット
FUNC. UNIT L1 S1 M1 D1 D2 M2 S2 L2
A15 A13 A11 A9 A7 A5 A3 A1
A14 A12 A10 A8 A6 A4 A2 A0
B15 B13 B11 B9 B7 B5 B3 B1
B14 B12 B10 B8 B6 B4 B2 B0
32
REGISTER FILE A
レジスタファイルA
DATA 1
データ1
20a
20b
22
DATA 2
データ2
REGISTER FILE B
レジスタファイルB
DATA MEMORY INTERFACE/CONTROL
データメモリインタフェース/制御
16
0 1 2 3 4 5 6 7
8 9
DATA ADDRESS 1
データアドレス1
DATA ADDRESS 2
データアドレス2
INTERNAL DATA MEMORY (BYTE ADDRESSABLE)
内部データメモリ(バイト単位のアクセス可能)

【図２７】



フロントページの続き

(72)発明者　レイド　イー．タトゲ
アメリカ合衆国　テキサス州リッチモンド，コッパーウッド　レーン　1707

(72)発明者　アラン　エル．デービス
アメリカ合衆国　テキサス州シュガーランド，マラナサ　ドライブ　3739

【外国語明細書】

Data Processing With Control Arrangement for Specifying Parallel and Sequential Execution of Program Instructions

Cross Reference to Related Applications

This application is related to coassigned application serial number 60/036,482 (TI docket number T-25311) filed contemporaneously herewith and incorporated herein by reference.

Field of the Invention

This invention relates generally data processing and, more particularly, to data processing with both parallel and sequential execution of program instructions.

Background of the Invention

Data processing systems and data processors are used in myriad applications which in turn have an impact on virtually every aspect of life. The utility of these myriad applications can ordinarily be enhanced by increasing the speed and throughput of the associated data processing systems and data processors.

One way to enhance speed and throughout is, where possible, to execute program instructions in parallel rather than in sequential fashion. One known approach in this regard is to utilize a special mode instruction

which specifies parallel or sequential execution of program instruction. Another known approach in this regard is to use a mask to specify null instructions in a parallel type packet. Another known solution is to perform data processing in a parallel mode only.

Although the aforementioned techniques are capable of improving speed and throughput, they are nevertheless undesirably difficult to implement and disadvantageously costly in terms of processing overhead.

It is therefore desirable to provide for parallel execution of program instructions in a manner which reduces the implementational difficulties and processing overhead associated with the above-described approaches.

An object of the present invention is to utilize a portion of a given program instruction to determine whether that instruction can be executed simultaneously with another program instruction.

Summary of the Invention

In general, and in a form of the present invention, a data processing device is provided which has circuitry for producing a set of instructions which include instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions. The data processing circuitry includes program execution circuitry connected to the producing circuitry for receiving the set of instructions and selectively executing simultaneously a plurality of the instructions in response to indicator portions.

In another form of the present invention, a method for operating a central processing unit (CPU) within a data processing device comprises the steps of: providing a set of instructions with respective instruction portions for indicating whether the respective instructions can be executed simultaneously; and determining from the instruction portions whether a plurality of the instructions can be executed simultaneously.

Other embodiments of the present invention will be evident from the description and drawings.

Brief Description of the Drawings

Fig. 1 is a block diagram of a data processing system according to the present invention;

Fig. 2 is a block diagram of a portion of the data processing circuitry of Fig. 1;

Fig. 3 illustrates the basic format of an instruction packet utilized in the present invention;

Fig. 4 is one example of an instruction packet according to the format of Fig. 3;

Fig. 5 illustrates the execution sequence defined by the instruction packet of Fig. 4;

Fig. 6 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 7 illustrates the execution sequence defined by the instruction packet of Fig. 6;

Fig. 8 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 9 illustrates the execution sequence defined by the instruction packet of Fig. 8;

Fig. 10 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 11 illustrates the execution sequence defined by the instruction packet of Fig. 10;

Fig. 12 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 13 illustrates the execution sequence defined by the instruction packet of Fig. 12;

Fig. 14 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 15 illustrates the execution sequence defined by the instruction packet of Fig. 14;

Fig. 16 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 17 illustrates the execution sequence defined by the instruction packet of Fig. 16;

Fig. 18 illustrates another example an instruction packet according to the format of Fig. 3;

Fig. 19 illustrates the execution sequence defined by the instruction packet of Fig. 18;

Figure 20 is a block diagram of a microprocessor which has an embodiment of the present invention;

Figure 21 is a block diagram of the execution units and register files of the microprocessor of Fig. 20;

Figure 22A is a chart which illustrates the processing phases of an instruction execution pipeline in the microprocessor of Fig. 20;

Figure 22B is a chart which illustrates the execution phases of the instruction execution pipeline in the microprocessor of Fig 20;

Figure 23 is a timing diagram which illustrates timing details of processing an instruction fetch packet during the processing phases of Fig. 22A and execution of the execution packet during the execution phases of Fig. 22B;

Figure 24 is a block diagram showing instruction dispatching in the microprocessor of Fig. 20;

Figure 25 illustrates the basic format of an instruction fetch packet for the microprocessor of Fig. 20; and

Figure 26 illustrates an 8-word fetch packet that is partially parallel.

Corresponding numerals and symbols in the different figures and tables refer to corresponding parts unless otherwise indicated.

Detailed Description

Fig. 1 is a block diagram of a data processing system 10 according to the present invention. The data processing system 10 includes data processing circuitry 11 and peripheral circuitries 13, 15, 17 and 19. In the exemplary embodiment of Fig. 1, the data processing circuitry 11 is connected to each of the peripheral circuitries 13, 15, 17 and 19 for transfer of information between data processing circuitry 11 and peripheral circuitries 13, 15, 17 and 19. However, and as will be apparent from the following description, a data processing system according to the present invention could include any quantity and type of peripheral circuitries and peripheral devices (such as peripherals 13,15, 17 and 19) interconnected among themselves and with data processing circuitry 11 in any manner heretofore or hereafter conceivable to workers in the art.

Fig. 2 illustrates a portion of one exemplary embodiment of the data processing circuitry 11 of Fig. 1. In Fig. 2, fetch circuitry 23 accesses memory 25 at an address specified by program counter 21 and causes an instruction packet at that address to be loaded into instruction register 27. Program execution circuitry 29 decodes and executes the instructions of the instruction packet held in instruction register 27.

Fig. 3 illustrates the basic format of the instruction packet fetched from memory 25. In the disclosed exemplary embodiment, an instruction packet includes four 32-bit instructions A, B, C and D. As shown in Fig. 3, instructions A, B, C and D are stored at consecutive addresses in memory 25. Thus, during normal sequential execution of program instructions, instruction A would be executed first, followed sequentially by instructions B, C and D.

Bit 0 of each instruction in Fig. 3 has been designated as a p-bit. The p-bits define how the instructions will be executed. The p-bits of the Fig. 3 instruction packet are inspected from left to right by the program execution circuitry 29. If the p-bit of a given instruction is equal to logic one, then the next sequential instruction in the packet is to be executed in parallel with

the first-mentioned instruction. Program execution circuitry 29 applies this rule until an instruction in the instruction packet is reached with a p-bit equal to logic 0.

If a given instruction has a p-bit of 0, then the next sequential instruction is executed sequentially after the given instruction (and after any instructions which are executed in parallel with the given instruction). The program execution circuitry 29 applies this rule until it reaches an instruction in the instruction packet with a p-bit of logic 1.

Figs. 4-19 provide application examples of the above-described p-bit rules.

Fig. 4 illustrates an instruction packet in which all p-bits are 0. Thus, instructions A-D are executed sequentially as shown in Fig. 5.

Fig. 6 illustrates an instruction packet in which the p-bits of instructions A, B and C are equal to 1, and the p-bit of instruction D is 0. Thus, instructions A, B, C and D are executed simultaneously, that is, in parallel as shown in Fig. 7.

In the instruction of Fig. 8, only the p-bit of instruction C is set to one, resulting in the execution sequence of Fig. 9, namely, instructions A and B are executed sequentially, followed by instructions C and D which are executed in parallel.

In the instruction packet of Fig. 10, only the p-bit of instruction B is set to one, resulting in the execution sequence shown in Fig. 11, namely instruction A is executed and then followed sequentially by the parallel execution of instructions B and C, which is then followed sequentially by execution of instruction D.

In the instruction packet of Fig. 12, the p-bits of instructions B and C are set to one, and the p-bits of instructions A and D are zero. This results in the instruction sequence shown in Fig. 13, namely instruction A is executed and is then sequentially followed by the parallel execution of instructions B, C and D.

In the instruction packet of Fig. 14, only the p-bit of instruction A is set to logic one, resulting in the execution sequence shown in Fig. 15,

namely instructions A and B are executed in parallel and then followed sequentially by the execution of instruction C and then the execution of instruction D.

In the instruction packet of Fig. 16, the p-bits of instructions A and C are set to one and the p-bits of instructions B and D are 0, resulting in the execution sequence illustrated in Fig. 17, namely the parallel execution of instructions A and B followed sequentially by the parallel execution of instructions C and D.

In the instruction packet of Fig. 18, the p-bits of instructions A and B are set to 1 and the p-bits of instructions C and D are 0. This results in the execution sequence illustrated in Fig. 19, namely instructions A, B and C are executed in parallel and then followed sequentially by execution of instruction D.

Because the instruction packet in the disclosed example includes 4 program instructions, the program compiler can always provide instruction D (the fourth instruction) with a p-bit of 0. The compiler determines the values of the remaining p-bits of instructions A, B and C based on the propriety of executing instructions A and B in parallel, the propriety of executing instructions B and C in parallel, and the propriety of executing instructions A, B and C in parallel. For example, if execution of instruction B requires a result provided by execution of instruction A, then the compiler would provide instruction A with a p-bit of 0 so that instruction B would be executed sequentially after instruction A. As another example, if instructions B and C access the same register, then the compiler would provide instruction B with a p-bit of 0 to ensure that instructions B and C are executed sequentially rather than in parallel.

Figure 20 is a block diagram of a microprocessor 1 which has an embodiment of the present invention. Microprocessor 1 is a VLIW digital signal processor ("DSP"). In the interest of clarity, Figure 20 only shows those portions of microprocessor 1 that are relevant to an understanding of an embodiment of the present invention. Details of general construction for DSPs are well known, and may be found readily elsewhere. For example,

U.S. Patent 5,072,418 issued to Frederick Boutaud, et al, describes a DSP in detail and is incorporated herein by reference. U.S. Patent 5,329,471 issued to Gary Swoboda, et al, describes in detail how to test and emulate a DSP and is incorporated herein by reference. Details of portions of microprocessor 1 relevant to an embodiment of the present invention are explained in sufficient detail hereinbelow, so as to enable one of ordinary skill in the microprocessor art to make and use the invention.

In microprocessor 1 there are shown a central processing unit (CPU) 10, data memory 22, program memory 23, peripherals 60 and an external memory interface (EMIF) with a direct memory access (DMA) 61. CPU 10 further has an instruction fetch/decode unit 10a-c , a plurality of execution units, including an arithmetic and load/store unit D1, a multiplier M1, an ALU/shifter unit S1, an arithmetic logic unit ("ALU") L1, a shared multiport register file 20a from which data are read and to which data are written. Decoded instructions are provided from the instruction fetch/decode unit 10a-c to the functional units D1, M1, S1, and L1 over various sets of control lines which are not shown. Data are provided to/from the register file 20a from/to to load/store units D1 over a first set of busses 32a, to multiplier M1 over a second set of busses 34a, to ALU/shifter unit S1 over a third set of busses 36a and to ALU L1 over a fourth set of busses 38a. Data are provided to/from the memory 22 from/to the load/store units D1 via a fifth set of busses 40a. Note that the entire data path described above is duplicated with register file 20b and execution units D2, M2, S2, and L2. Instructions are fetched by fetch unit 10a from instruction memory 23 over a set of busses 41. Emulation unit 50 provides access to the internal operation of integrated circuit 1 which can be controlled by an external test system 51.

Note that the memory 22 and memory 23 are shown in Figure 20 to be a part of a microprocessor 1 integrated circuit, the extent of which is represented by the box 42. The memories 22-23 could just as well be external to the microprocessor 1 integrated circuit 42, or part of it could reside on the integrated circuit 42 and part of it be external to the

integrated circuit 42. Also, an alternate number of execution units can be used.

When microprocessor 1 is incorporated in a data processing system, additional memory or peripherals may be connected to microprocessor 1, as illustrated in Figure 1. For example, Random Access Memory (RAM) 70, a Read Only Memory (ROM) 71 and a Disk 72 are shown connected via an external bus 73. Bus 73 is connected to the External Memory Interface (EMIF) which is part of functional block 61 within microprocessor 42. A Direct Memory Access (DMA) controller is also included within block 61. The DMA controller is generally used to move data between memory and peripherals within microprocessor 1 and memory and peripherals which are external to microprocessor 1.

Several example systems which can benefit from aspects of the present invention are described in U.S. Patent 5,072,418, which was incorporated by reference herein, particularly with reference to Figures 2-18 of U.S. Patent 5,072,418. A microprocessor incorporating an aspect of the present invention to improve performance or reduce cost can be used to further improve the systems described in U.S. Patent 5,072,418. Such systems include, but are not limited to, industrial process controls, automotive vehicle systems, motor controls, robotic control systems, satellite telecommunication systems, echo canceling systems, modems, video imaging systems, speech recognition systems, vocoder-modem systems with encryption, and such.

A description of various architectural features of the microprocessor of Fig. 20 is provided in coassigned application serial number 60/036,482 (TI docket number T-25311). A description of a complete set of instructions for the microprocessor of Fig. 20 is also provided in coassigned application serial number 60/036,487 (TI docket number T-25311).

Figure 21 is a block diagram of the execution units and register files of the microprocessor of Fig. 20 and shows a more detailed view of the buses connecting the various functional blocks. In this figure, all data busses are 32 bits wide, unless otherwise noted. Bus 40a has an address bus DA1

which is driven by mux 200a. This allows an address generated by either load/store unit D1 or D2 to provide an address for loads or stores for register file 20a. Data Bus LD1 loads data from an address in memory 22 specified by address bus DA1 to a register in load unit D1. Unit D1 may manipulate the data provided prior to storing it in register file 20a. Likewise, data bus ST1 stores data from register file 20a to memory 22. Load/store unit D1 performs the following operations: 32-bit add, subtract, linear and circular address calculations. Load/store unit D2 operates similarly to unit D1, with the assistance of mux 200b for selecting an address.

ALU unit L1 performs the following types of operations: 32/40 bit arithmetic and compare operations; left most 1, 0, bit counting for 32 bits; normalization count for 32 and 40 bits; and logical operations. ALU L1 has input src1 for a 32 bit source operand and input src2 for a second 32 bit source operand. Input msb_src is an 8 bit value used to form 40 bit source operands. ALU L1 has an output dst for a 32 bit destination operands. Output msb_dst is an 8 bit value used to form 40 bit destination operands. Two 32 bit registers in register file 20a are concatenated to hold a 40 bit operand. Mux 211 is connected to input src1 and allows a 32 bit operand to be obtained from register file 20a via bus 38a or from register file 20b via bus 210. Mux 212 is connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 38a or from register file 20b via bus 210. ALU unit L2 operates similarly to unit L1.

ALU/shifter unit S1 performs the following types of operations: 32 bit arithmetic operations; 32/40 bit shifts and 32 bit bit-field operations; 32 bit logical operations; branching; and constant generation. ALU S1 has input src1 for a 32 bit source operand and input src2 for a second 32 bit source operand. Input msb_src is an 8 bit value used to form 40 bit source operands. ALU S1 has an output dst for a 32 bit destination operands. Output msb_dst is an 8 bit value used to form 40 bit destination operands. Mux 213 is connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 36a or from register file 20b via bus

210. ALU unit S2 operates similarly to unit S1, but can additionally perform register transfers to/from the control register file 102.

Multiplier M1 performs 16 x 16 multiplies. Multiplier M1 has input src1 for a 32 bit source operand and input src2 for a 32 bit source operand. ALU S1 has an output dst for a 32 bit destination operands. Mux 214 is connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 34a or from register file 20b via bus 210. Multiplier M2 operates similarly to multiplier M1.

Figure 22A is a chart which illustrates the processing phases of an instruction execution pipeline in the microprocessor of Fig. 1. Each phase corresponds roughly to a clock cycle of a system clock. For example, if microprocessor 1 is being operated at 200 MHz, then each phase is nominally 5 Ns. However, in a phase where data is expected from a memory or peripheral, such as RAM 70, the pipeline will stall if the data is not ready when expected. When stalled, a given pipeline phase will exist for a number of system clock cycles.

In Figure 22A, the first phase of processing an instruction is to generate the program address in phase PG. This is done by loading a program fetch counter PFC which is located in control register file 102. During the second instruction processing phase PS, an address of an instruction fetch packet is sent to program memory 23 via a program address bus PADDR which is part of bus 41. The third phase PW is a wait phase to allow for access time in memory 23. During the fourth phase PR, a program fetch packet is available from program memory 23 via data bus PDATA_I which is part of bus 41. During the fifth processing phase DP, instruction parallelism is detected and instructions that can be executed are dispatched to the appropriate functional units. This aspect of pipeline operation will be described in more detail in later paragraphs. During the sixth processing phase DC, executable instructions are decoded and control signals are generated to control the various data paths and functional units.

Figure 22B is a chart which illustrates the execution phases of the instruction execution pipeline in the microprocessor of Fig 1. During the first execution phase E1, single cycle instructions, referred to as "ISC", and branch instructions, referred to as "BR", are completed. A designated execution unit performs the operations indicated in Fig. 22B as directed by control circuitry 100. During the second execution phase E2, the following types of instructions are completed by designated execution units under control of control circuitry 100: integer multiply (IMPY), program store instructions (STP), and data store instructions (STD). During the third execution phase E3, execution of load data instructions (LD) continues by latching data from the data memory system (DMS), as indicated. During execution phase E4, the data latched in E3 is transferred to a data input register DDATA_I in execution unit D1 or D2. During execution phase E5, the LD instruction is completed by manipulating the data in register DDATA_I and writing the manipulated data to a specified register in register file 20a or 20b.

Figure 23 is a timing diagram which illustrates timing details of processing an instruction fetch packet during the processing phases of Fig. 22A and execution of the execution packet during the execution phases of Fig. 22B. Note that a pipe stall is illustrated in phase PW due to a program memory ready signal PRDY being low in phase PS, and a second pipe stall in phase E3 due a data memory ready signal DRDY being low in phase E2.

Figure 24 is a block diagram showing instruction dispatching in the microprocessor of Fig. 20. In this embodiment, an instruction fetch packet contains eight instructions. Instruction fetch packet 1710 is dispatched and decoded to eight execution units as illustrated. Fetch packet 1720 contains a branch instruction 1725. Instruction fetch packet 1730 contains three instruction execute packets. The first execute packet contains two instructions, ZERO-SHL, which will begin processing in the first delay slot of branch instruction 1725. The second execute packet contains four instruction, ADD-SUB-STW-STW, which will begin processing in the second delay slot of branch instruction 1725. The third execute packet

contains two instructions, ADDK-BR, which will begin processing in the third delay slot of branch instruction 1725.

Parallel Operations

Instructions are always fetched eight at a time. This constitutes a fetch packet. The basic format of a fetch packet is shown in Figure 25. The execution grouping of the fetch packet is specified by the *p*-bit, bit zero, of each instruction. Fetch packets are 8-word aligned.

The p bit controls the parallel execution of instructions. The *p*-bits are scanned from left to right (lower to higher address). If the p bit of instruction *i* is 1, then instruction *i* + 1 is to be executed in parallel with (in the same cycle as) instruction *i*. If the *p*-bit of instruction *i* is 0, then instruction *i* + 1 is executed in the cycle after instruction I. All instructions executing in parallel constitute an execute packet. An execute packet can contain up to eight instructions. All instructions in an execute packet must use a unique functional unit.

An execute packet cannot cross an 8-word boundary. Therefore, the last *p*-bit in a fetch packet is always set to 0, and each fetch packet starts a new execute packet. As discussed with regard to Figures 4-19, there are three types of *p*-bit patterns for fetch packets. These three *p*-bit patterns result in the following execution sequences for the eight instructions: Fully serial, Fully parallel, Partially serial.

Example Parallel Code

The | | characters signify that an instruction is to execute in parallel with the previous instruction. In the fetch packet of Figure 26, the code would be represented as this:

```
        instruction A

        instruction B

        instruction C
||      instruction D
||      instruction E
```

```
        instruction F
||      instruction G
||      instruction H
```

Branching Into the Middle of an Execute Packet

If a branch into the middle of an execution packet occurs, all instructions at lower addresses are ignored. In the example in Figure 26, if a branch to the address containing instruction D occurs, then only D and E will execute. Even though instruction C is in the same execute packet, it is ignored. Instructions A and B are also ignored because they are in earlier execute packets.

Resource Constraints

No two instructions within the same execute packet can use the same resources. Also, no two instructions can write to the same register during the same cycle. The following sections describe each of the resources an instruction can use.

Functional Units

Two instructions using the same functional unit cannot be issued in the same execute packet.

The following execute packet is invalid:

```
        ADD S1    A0, A1, A2   ; \  .S1 is used for
||      SHR .S1   A3, 16, A4   ; /  both instructions
```

The following execute packet is valid:

```
        ADD L1    A0, A1, A2   ; \ Two different functional
||      SHR .S1   A3, 15, A4   ; /   units are used
```

Cross Paths (1X and 2X)

One unit (either a .S, .L, or .M) per data path, per execute packet, can read a source operand from its opposite register file via the cross paths (1X and 2X). For example, .S1 can read both operands from the A register file, or one operand from the B register file using the 1X cross path. This is denoted by an X following the unit name.

Two instructions using the same X cross path between register files cannot be issued in the same execute packet since there is only one path from A to B and one path from B to A.

The following execute packet is invalid:

```
        ADD.L1X   A0,B1,A1   ; \ 1X cross path is used
||      MPY.M1X   A4,B4,A5   ; / for both instructions
```

The following execute packet is valid:

```
        ADD.L1X   A0,B1,A1   ; \ Instructions use the 1x and
||      MPY.M2X   A4,B4,132  ; /     2x cross paths
```

The operand will come from a register file opposite of the destination if the x bit in the instruction field is set.

Load and Store Path

Loads and stores can use an address pointer from one register file while loading to or storing from the other register file. Two loads and/or stores using an address pointer from the same register file cannot be issued in the same execute packet.

The following execute packet is invalid:

```
        LDW D1    *A0,A1     ; \ Address registers from the same
||      LDW.D1    *A2,B2     ; /    register file
```

The following execute packet is valid:

```
        LDW D1    *A0,A1     ; \ Address registers from different
||      LDW.D2    *B0,B2     ; /  register files
```

Two loads and/or stores loading to and/or storing from the same register file cannot be issued in the same execute packet.

The following execute packet is invalid:

```
        LDW D1    *A4,A5     ; \ Loading to and storing from the
||      STW.D2    A6,*B4     ; /      same register file
```

The following execute packet is valid:

```
        LDW D1    *A4,B5     ; \ Loading to, and storing from
```

```
||   STW.D2    A6,*B4      ; /  different register files
```

Long Paths

Only one long result may be written per cycle on each side of the register file. Because the S and .L units share a read register port for long source operands and a write register port for long results, only one may be issued per side in an execute packet.

The following execute packet is invalid:

```
     ADD.L1    A5:A4,A1,A3 :A2   ; \ Two long writes
||   SHL.S1    A8, A9, A7 :A6    ; /  on A register file
```

The following execute packet is valid:

```
     ADD.L1    A5:A4,A1,A3 :A2   ; \ One long write for
||   SHL.S2    B8, B9, B7 :B6    ; /  each register file
```

Because the .L and .S units share their long read port with the store port, operations that read a long value cannot be issued on the .L and/or .S units in the same execute packet as a store.

The following execute packet is invalid:

```
     ADD.L1    A5:A4,A1,A3 :A2   ; \ Long read operation and a
||   STW.D1    A8, *A9           ; /  store
```

The following execute packet is valid:

```
     ADD.L1    A4, A1, A3 :A2    ; \ No long read with
||   STW.D1    AB,.*A9           ; /  with the store
```

Register Reads

More than four reads of the same register cannot occur on the same cycle. Conditional registers are not included in this count.

The following code sequence is invalid:

```
     MPY       .M1   A1,A1,A4    ; five reads of register A1
||   ADD       .L1   A1,A1,A5
||   SUB       .D1   A1,A2,A3
```

Whereas this code sequence is valid:

```
     MPY       .M1   A1,A1,A4    ; only four reads of A1
```

```
||  [A1]  ADD     .L1   A0,A1,A5
          SUB     .D1   A1,A2,A3
```

Register Writes

Multiple writes to the same register on the same cycle can occur if instructions with different latencies writing to the same register are issued on different cycles. For example, an MPY issued on cycle i followed by an ADD on cycle i+1 cannot write to the same register since both instructions will write a result on cycle i+1. Therefore, the following code sequence is invalid:

```
MPY     .M1   A0,A1,A2
ADD     .L1   A4,A5,A2
```

Detectability of Write Conflicts

The following sequence of execute packets shows different multiple write conflicts. For example, the ADD and SUB in execute packet L1 write to the same register. This conflict is easily detectable.

```
L1:           ADD.L2   B5,B6,B7   ; detectable, conflict
||            SUB.S2   B8,B9,B7
L2:           MPY.M2   B0,B1,B2   ; \ not detectable
L3            ADD.L2   B3,B4,B2   ; /
L4:  [!B0]    ADD.L2   B5,B6,B7   ; detectable, no conflict
||   [B0]     SUB.S2   B8, B9 B7
L5:  [!B1]    ADD.L2   B5,B6,B7   ; \ not detectable
||   [B0]     SUB.S2   B8,B9,B7   ; /
```

The MPY in packet L2 and the ADD in packet L3 might both write to B2 simultaneously; however, if a branch instruction causes the execute packet after L2 to be something other than L3, this would not be a conflict. Thus, the potential conflict in L2 and L3 might not be detected by the assembler. The instructions in L4 do not constitute a write conflict because they are mutually exclusive. In contrast, because it is not obvious that the instructions in L5 are mutually exclusive, the assembler cannot determine a conflict. If the pipeline does receive commands to perform multiple writes to the same register, the result is undefined.

Although exemplary embodiments of the present invention are described above, this does not limit the scope of the invention, which can be practiced in a variety of embodiments.

WHAT IS CLAIMED IS:

1. A data processing system, comprising:

data processing circuitry having circuitry for producing a set of instructions which include respective instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions; and

said data processing circuitry including program execution circuitry connected to said producing circuitry for receiving the set of instructions and selectively responsive to said instruction portions for executing simultaneously a plurality of said instructions indicated by said instruction portions.

2. A method of processing program instructions in a data processing system, comprising the steps of:

providing a set of instructions with respective instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions; and

determining from said instruction portions whether a plurality of said instructions can be executed simultaneously.

3. A method of compiling a program for execution by a data processing system, comprising the steps of:

determining whether a first program instruction can be executed simultaneously with a second program instruction that immediately sequentially follows the first program instruction in program; and

providing the first instruction with an instruction portion that indicates whether the first instruction can be executed simultaneously with the second instruction.

FIG. 1

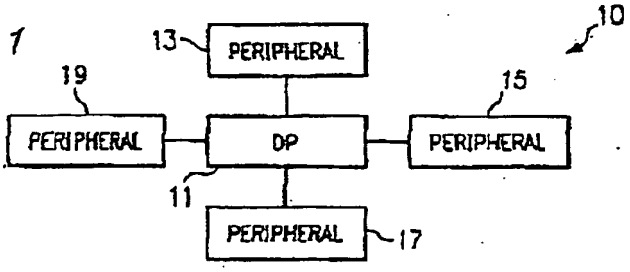


FIG. 2

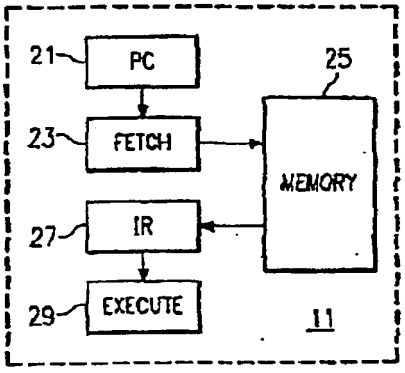


FIG. 5

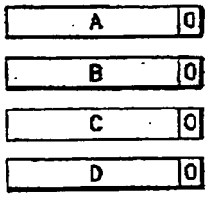


FIG. 3

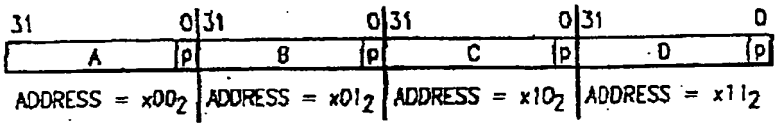


FIG. 4

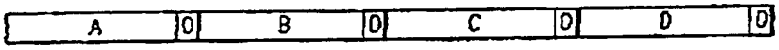


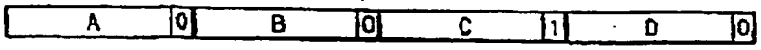
FIG. 6

FIG. 7

2

11-23309
2 of 9/10

FIG. 8



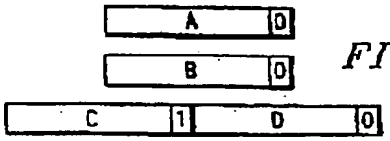
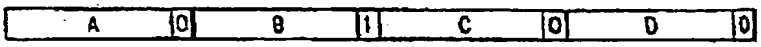


FIG. 9

FIG. 10



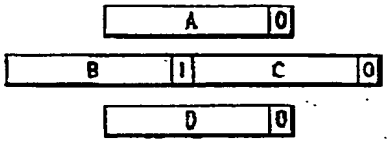


FIG. 11

FIG. 12

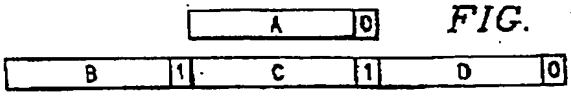
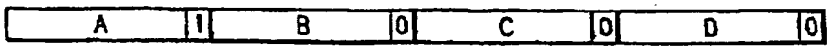


FIG. 13

3

11-25309
3 of 8/10

FIG. 14



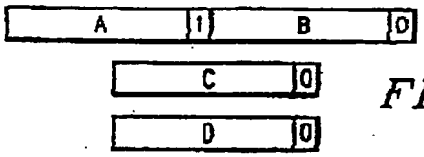
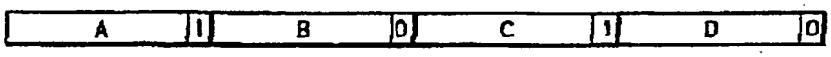


FIG. 15

FIG. 16



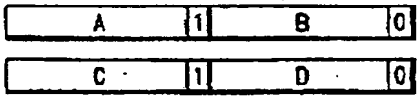


FIG. 17

FIG. 18

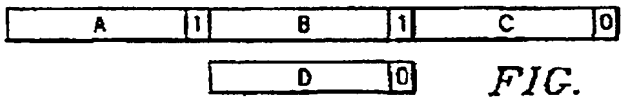
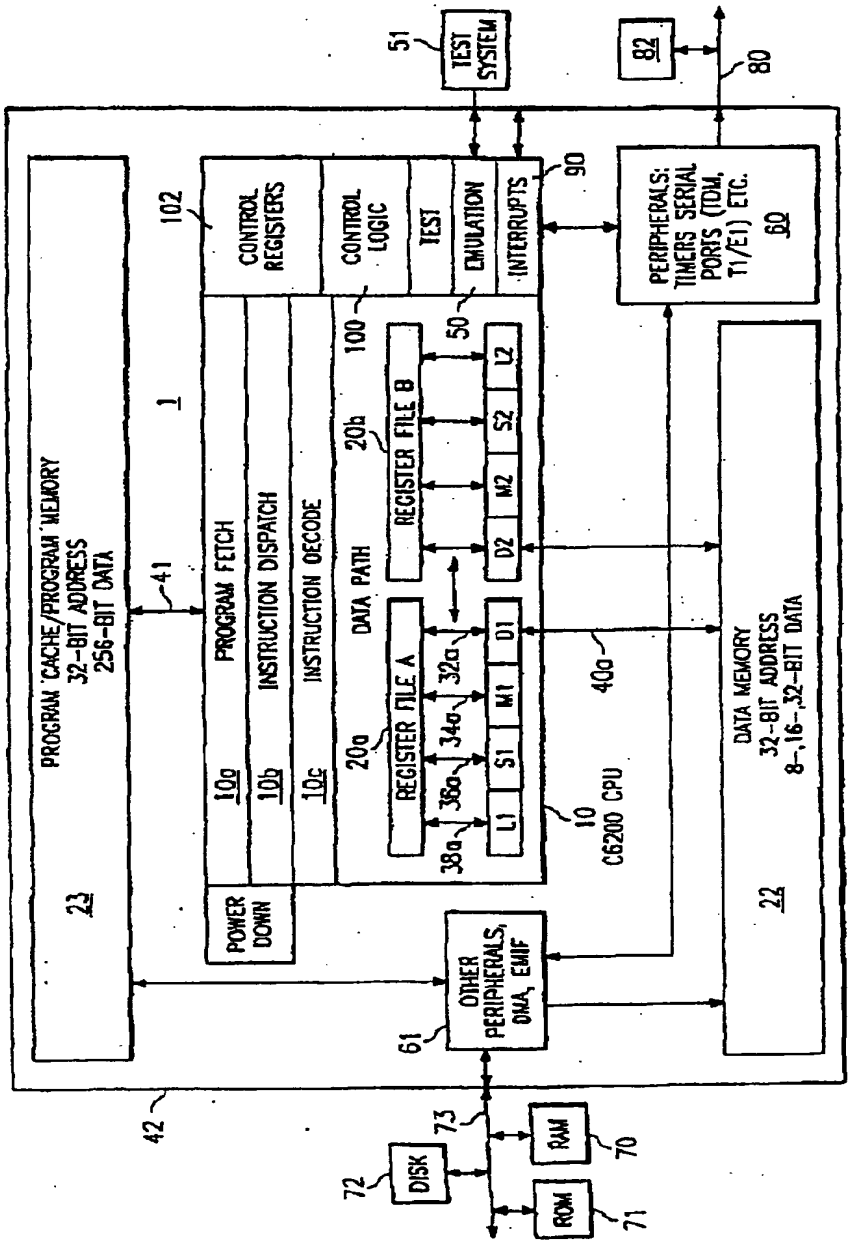


FIG. 19

4

FIG. 20



5

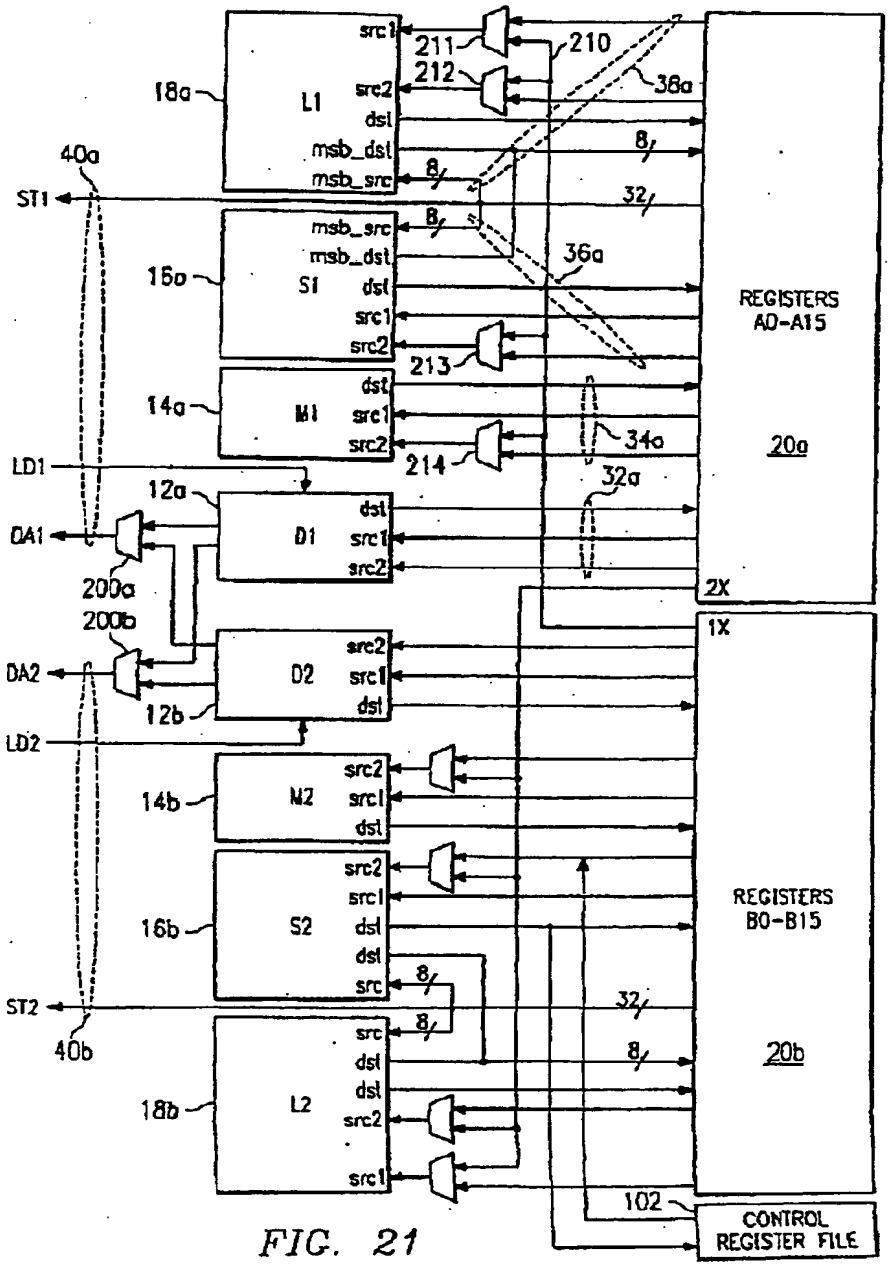


FIG. 21

FIG. 22A

Pipeline Phase	Symbol	During this Entire Phase	By the End of This Phase
Program Address Generate	PG		◻ Except when a program store is in E1, PFC and PADDR and latched to their next values: either the next 8-word boundary after the current PFC or a branch target address. ◻ When a program store is in E1, the PFC remains unchanged and PADDR takes the value from the program store instruction.
Program Address Send	PS	◻ PFC valid on PADDR. ◻ If a fetch packet fetch needed, PRS set active.	◻ If PRS active, PMS latches PADDR.
Program Wait	PW	◻ If the fetch packet indicated by the last PRS is necessary, PDS is asserted.	◻ If PDS is asserted, PMS latches the last requested fetch packet to be driven during PR.
Program Data Receive	PR	◻ If PDS active during the associated PW, PMS drives requested fetch packet onto PDATA_1. Otherwise the previous fetch packet driven on to PDATA_1 is maintained.	◻ If it can be determined that the fetch packet requested by an active PDS in PW is needed, CPU latches this fetch packet on PDATA_1. Otherwise, this latch maintains state.
Dispatch	DP		◻ Parallelism of the fetch packet detected. ◻ Instructions in the next requested execute packet in the fetch packet dispatched to the appropriate functional units. ◻ Detection of the software breakpoint code in the CREG field detected. ◻ Multi-cycle NOP, NOP, SWI, and IDLE decoded because they are not passed to individual functional units. ◻ PC of the execute packet in DP latched at the end of the cycle as PCDC.
Decode	DC	◻ PCDC valid.	◻ Control signals for actions during execute decoded.

FIG. 22B

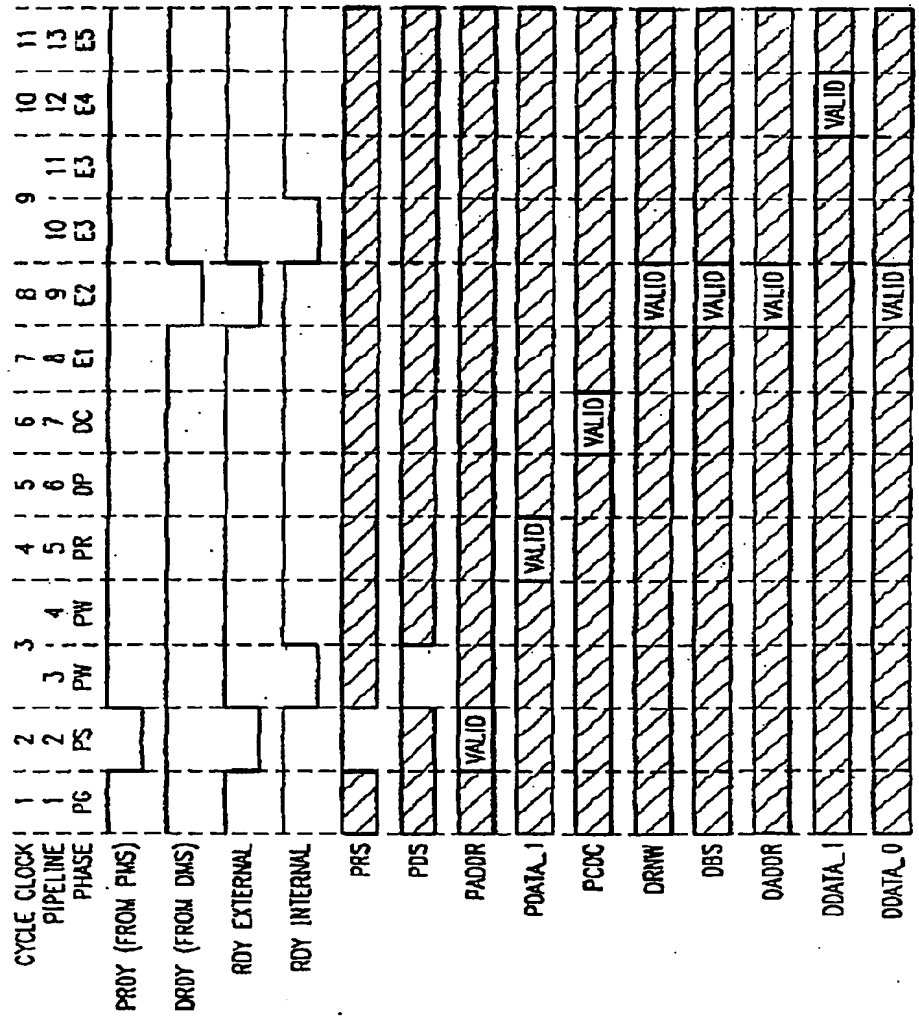
Pipeline Phase	Symbol	During this Phase	By the End of This Phase	CPU has Completed Types
Execute 1	E1	◻ All Types: Registers read.	◻ Type ISS: Results written to the register file. ◻ Type ISS: Status written to the control register file. ◻ Type LD and ST: Address modifications written to the register file. ◻ Type LD and ST: DADDR, DBS, and DRNW latched for next phase. ◻ Type ST: DDATA_0 latched for next phase. ◻ Type BRANCH: PFC and PADDR latches the target address. ◻ Type STP: PADDR latches the destination address if no fetch is pending. PWS latched as active for next phase. ◻ Types LD, ST, STP, IMPY: Intermediate states latched.	◻ ISC. ◻ BR.
Execute 2	E2	◻ Types LD and ST: DBS active and DADDR valid. ◻ Type LD: DRNW active. ◻ Type ST: DRNW inactive and DDATA_0 valid.	◻ Type IMPY: Results written to the register file.[3] ◻ Type IMPY: Status written to the control register file. ◻ Types LD and ST: DMS latches ADDR. ◻ Type ST: DMS latches DDATA_0. ◻ Type STP: PMS latches PDATA_0. ◻ Type LD: Intermediate states latched.	◻ IMPY. ◻ STD. ◻ STP.
Execute 3	E3		◻ Type LD: DMS latches data requested during the last cycle. ◻ Types LD: Intermediate states latched.	
Execute 4	E4	◻ Type LD: DMS drives on DDATA_1 data requested during associated E2.	◻ Type LD: DDATA_1 latched. ◻ Types LD: Intermediate states latched.	
Execute 5	E5		◻ Type LD: DDATA_1 right-aligned, either sign-extended or zero-filled, and written to the register file.	◻ LD.

11-23505
7 of 8/D

特開平10-232779

8

FIG. 23


CYCLE CLOCK
PIPELINE PHASE
PRDY (FROM PMS)
DRDY (FROM DMS)
RDY EXTERNAL
RDY INTERNAL
PRS
PDS
PADDR
PDATA_I
PCDC
DRNW
DBS
DADDR
DDATA_I
DDATA_O
VALID

9

TI-25309
9 of 9/0

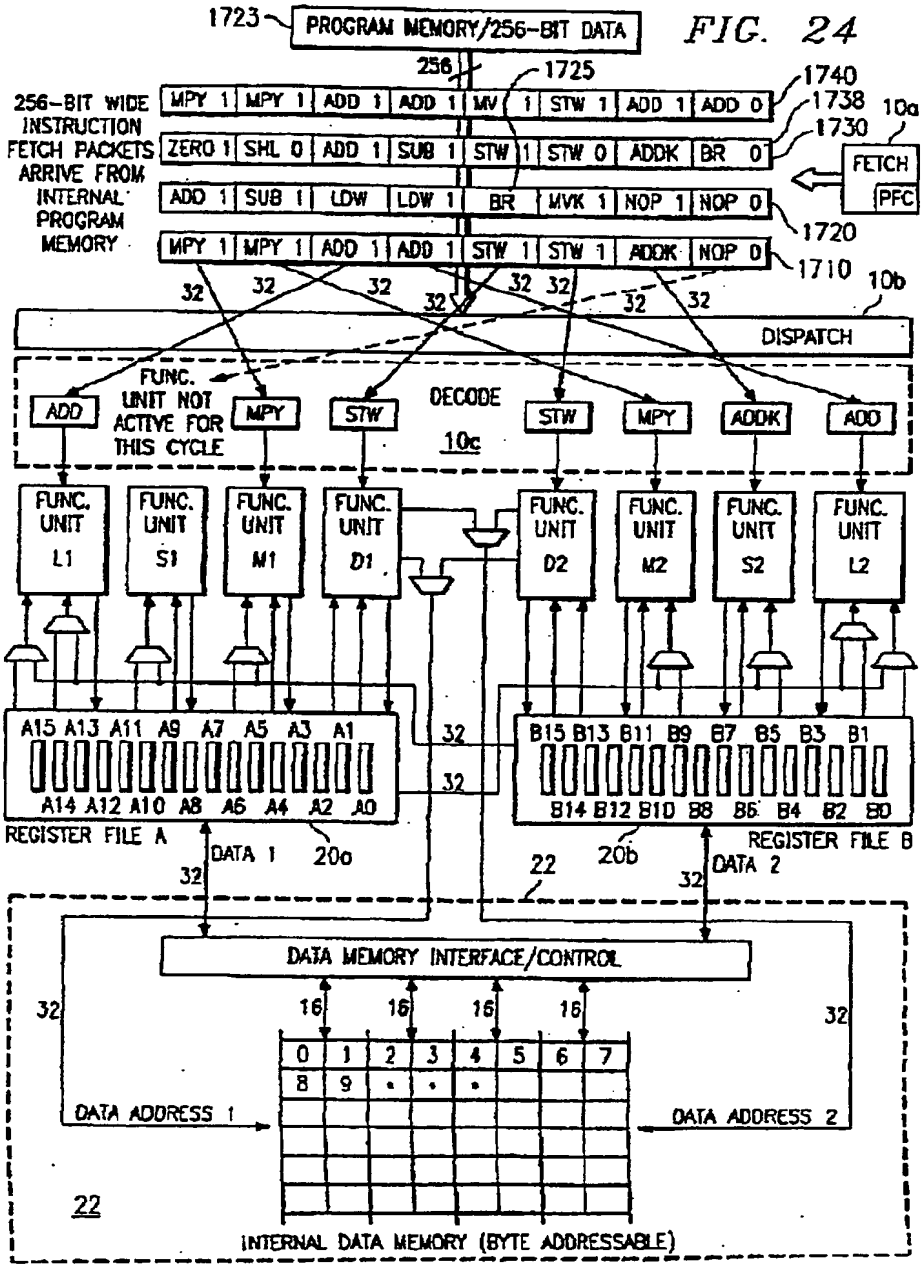
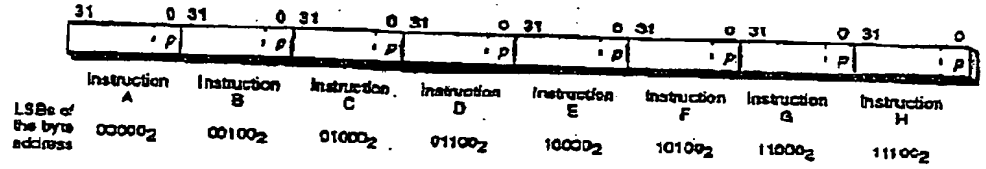


FIG. 24

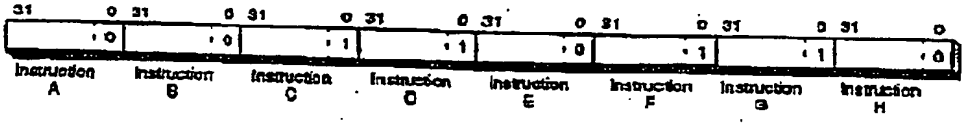
TI-25309 10
1/10

Fig. 25

~~Figure 25 The Basic Format of a Fetch Packet~~



This *p*-bit pattern:



results in this execution sequence:

Cycle	Instructions		
1	A		
2	B		
3	C	D	E
4	F	G	H

Note: Instructions C, D, and E do not use any of the same functional units, cross paths, or other data path resources. This is also true for instructions F, G, and H.

Figure 26

1. Abstract

A data processing system 10 includes data processing circuitry 11 having circuitry (21, 23 and 25) for producing a set of instructions which include respective instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions. Program execution circuitry (29) receives the set of instructions and is selectively responsive to the instructive portions for executing simultaneously a plurality of the instructions indicated by the instruction portions.

2 Representative Drawing

Fig 2